Be Powered by Rexy Embedded™

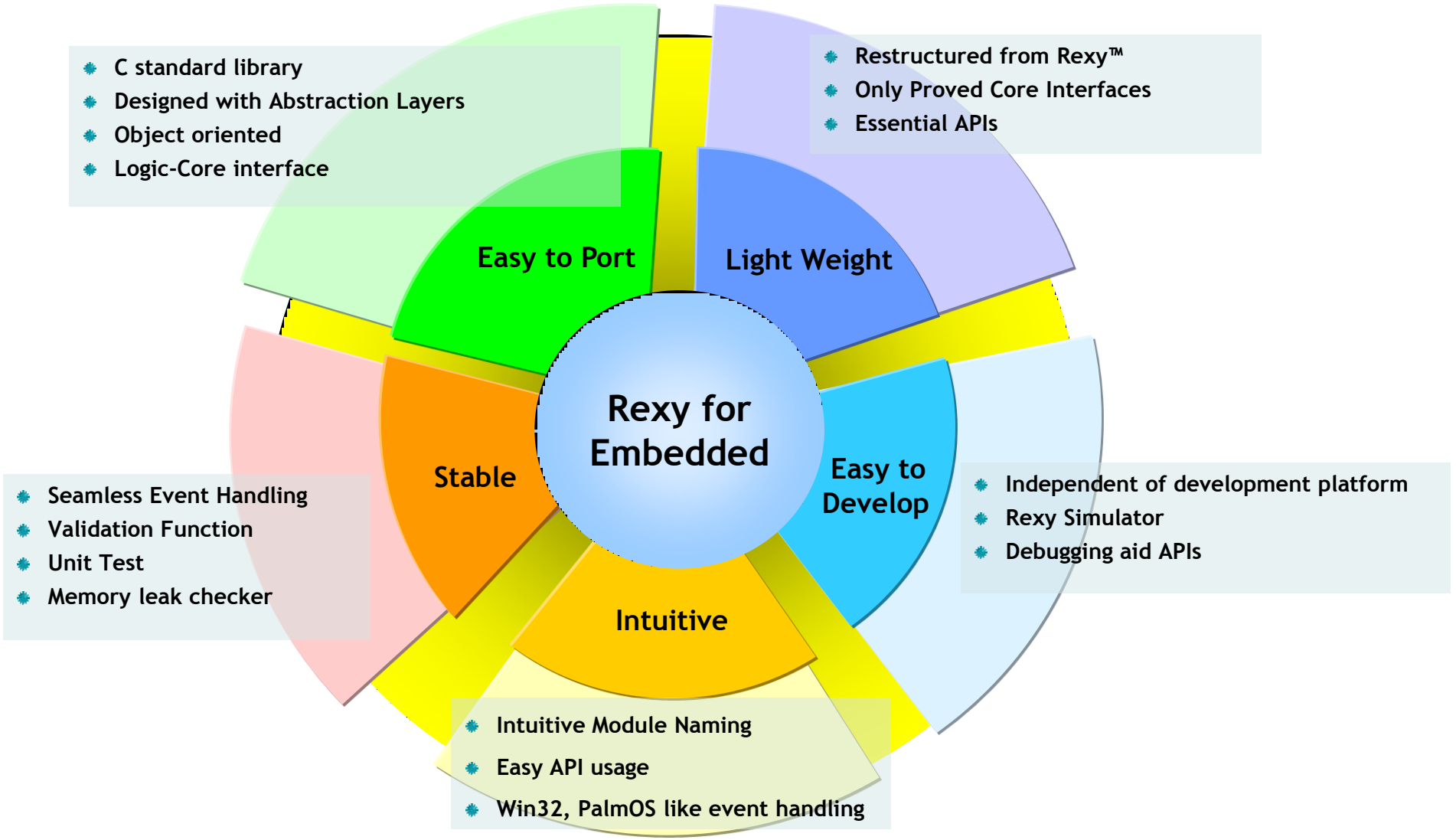
INDEX

1. OVERVIEW

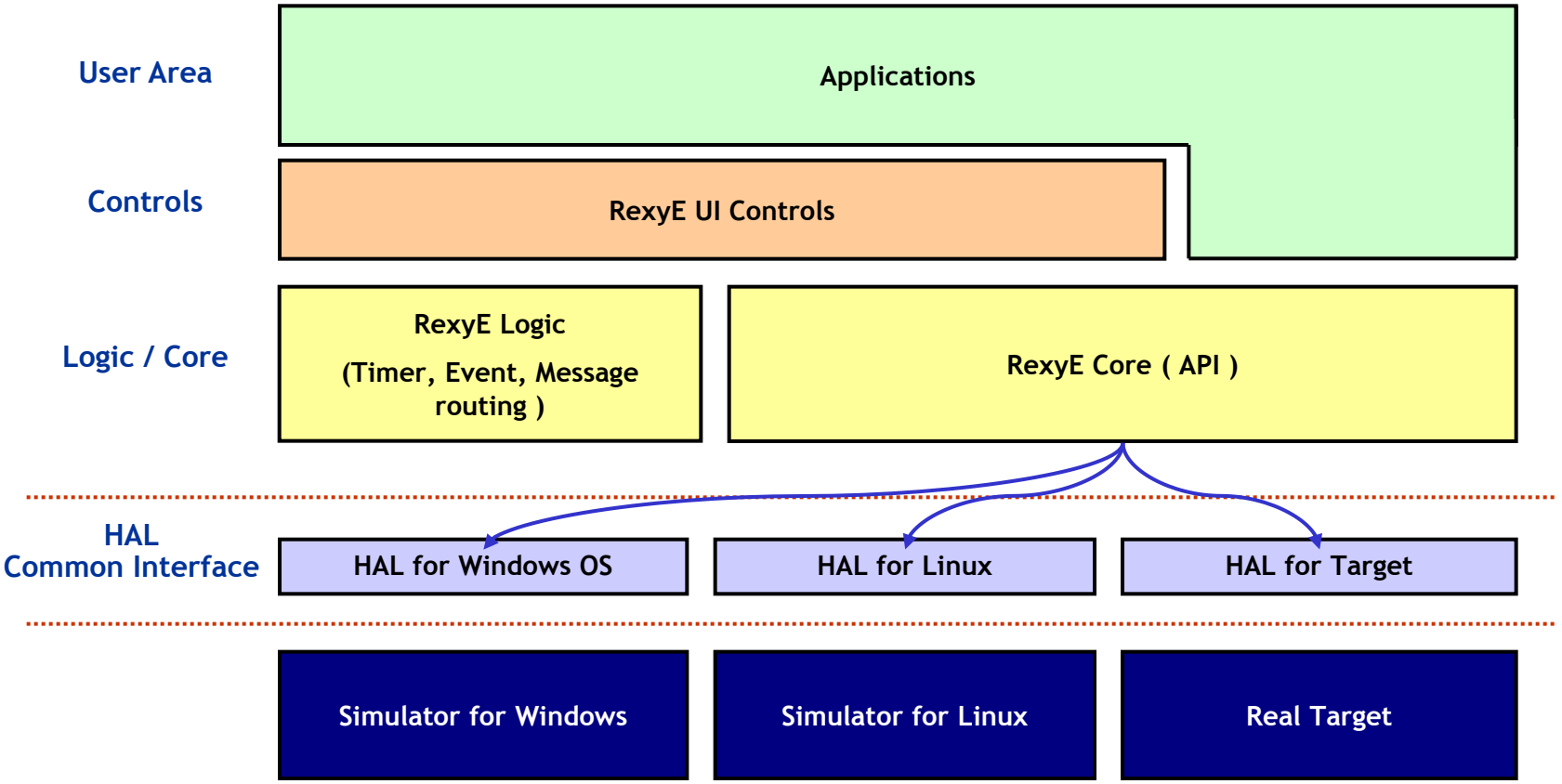
1. What is Rexy Embedded™?
2. Concept
3. Structure
4. Minimal Requirement

Rexy Embedded™ !

- **Best Optimized Graphical User Interface(GUI) Development Platform for Embedded Devices.**
- Supports powerful GDI, messaging, window, and other instinctive APIs.
- Operate on most of embedded OS (RTOS, Rex , µC/OS-II, iTron, Embedded Linux, Windows CE …)
- Simulator, Font Converter and Image Converter for Developer !



- Structure Diagram



Operation Environment

Items	Details
OS	RTOS, µC/OS-II, iTron, Linux, WinCE..
CPU	ARM Device, x86, m68k etc.
LCD	1, 2, 8, 12, 15, 16, 24 bit plain LCD

Memory Size

(unit : KB)

Items	X86		ARM	
	Code	Data	Code	Data
Core	30	21	23	19
Logic	1	0.8	1	0.4
Controls (Built-in)	20	2	17	0.2
HAL	5	461	1	153
Total	**56**	**484.8**	**42**	**172.6**

Development Environment

Items	Details
OS	Win95 or higher, Linux (gtk 2.0+)
CPU	Over P3 500MHz
Development S/W	MS Visual Studio
	VC Toolkit 2005 + (Eclipse, Visual Slick Editor, Emacs, other editors) ..
	GCC + (Eclipse, Visual Slick Editor, Emacs, other editors) ..

2. TECHNICAL HIGHLIGHTS

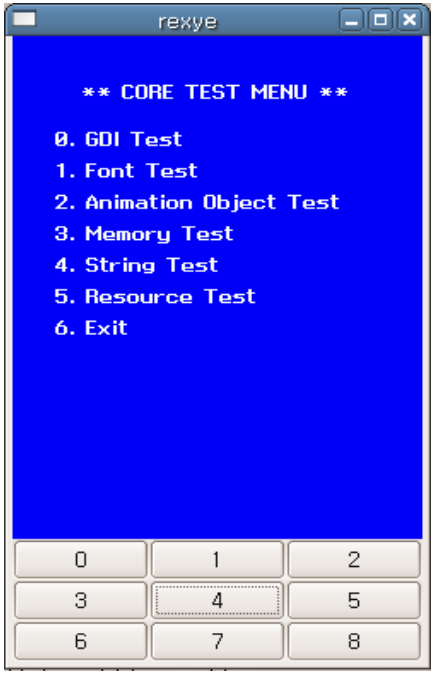
1. Easy Porting
2. Light Weight
3. Seamless Message Handling
4. Internationalization
5. Video Buffer
6. Useful Tools
   - 6.1 Simulator
   - 6.2 Font Converter
   - 6.3 Resource Editor
7. Comparing with other GUI libraries

Porting is Easy !

- Rexy for Embedded Supports HAL (Hardware Abstraction Layer)
- Only thing that you have to do is just adjusting drivers and a few settings

Windows

Linux



Real Target

It’s simple and light !

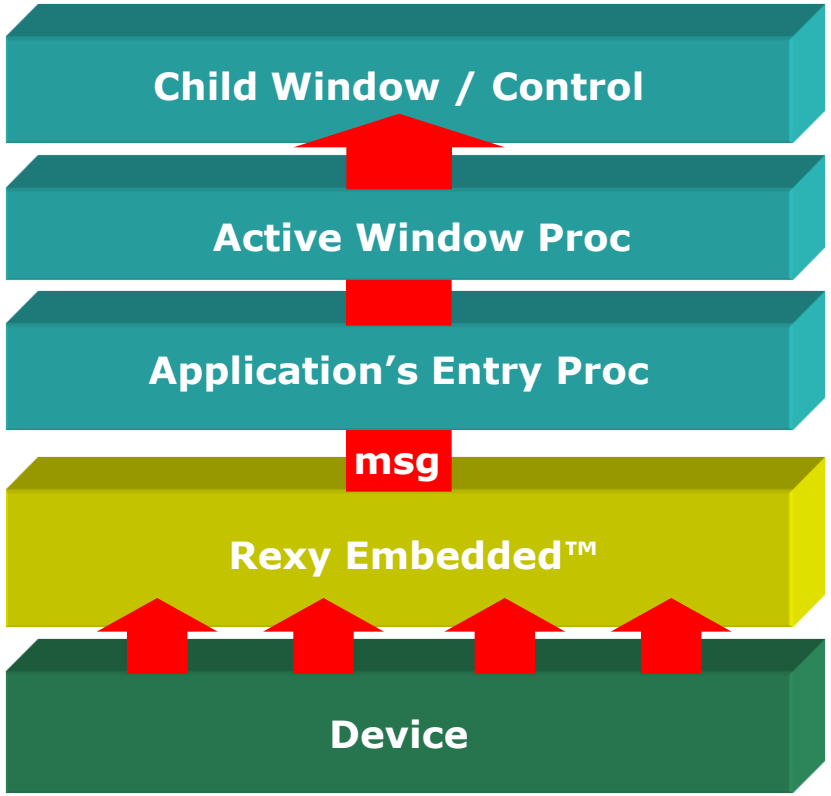
- Event driven structure
- Fast run time speed with C language

- The size of executable binary is only 40 KB ! (Fonts and resources are excluded)
- Only 20 kb of static memory can execute Rexy Embedded™

GUI Libraries	Rexy Embedded™	Zinc	microGUI	picoGUI	Qtopia
Executable Binary Size	40 KB	350 ~ 750 KB	Under 1 MB	A few hundred KB	Phone & PDA Edition -Flash : 8~16MB -SDRAM :16~32MB

You can get the stability through seamless message handling !

- Rexy Embedded™ hands over messages to current application after standardizing them
- Every messages has a tag pointing its own designated window
- You don’t have to forward messages to its sub-message handlers manually, for Rexy routes them automatically
- However, you can interfere message routing process to achieve even complicated and exceptional scenarios by simply controlling returning value of each event handler



You don’t have to worry about the internationalization !

- Supports add-in APIs to get information and modify for each encoding type.
- Supports multi-lingual string.

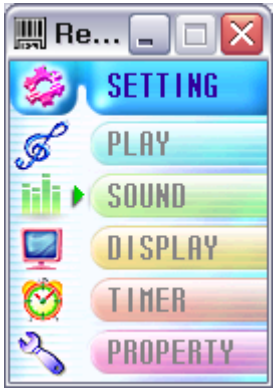
TYPE	SUPPORTED LIST
Character set & Encoding	ISO8859-1~ISO8859-16
	GB2312
	GBK
	GB18030-0
	BIG5
	UNICODE
	UTF-8
	Shift-JIS
	EUC

Video buffer for tidy GUI !

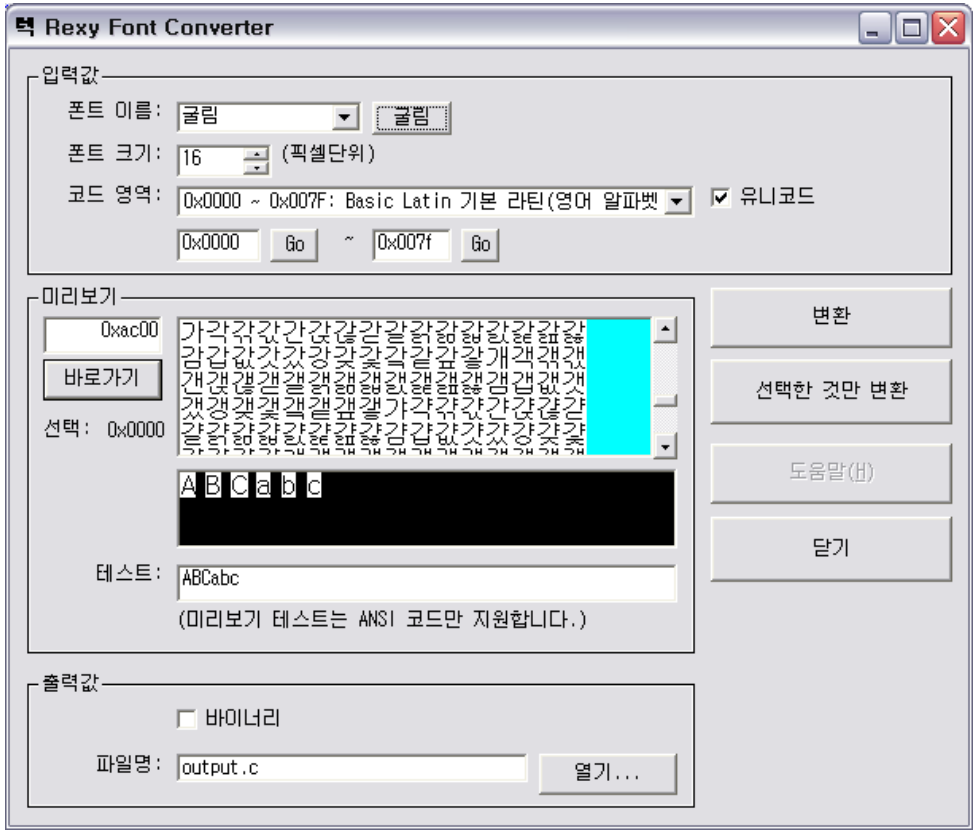
- Rexy Embedded™ GDI API draws image on video buffer so that it can update screen at once.
  It will make your GUI nice and tidy.

Develop your codes on a smart simulator !

- Same Environment equal to target platform in win32 platform
- Support MS Visual C++
- Support Debug Message Window
- Support Double-sized Window



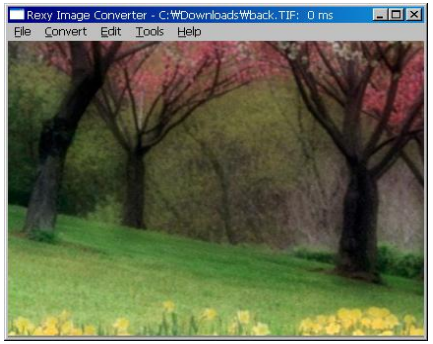
You cannot make your own fonts faster than this !

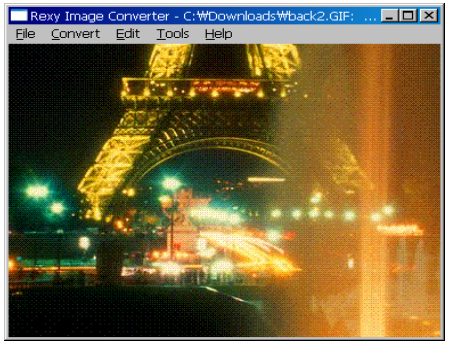


- Extract and convert any desired fonts
- Unicode fonts also can be created
- Preview feature
- Supports source code output & binary output

Image converter can convert more than 29 types of image !

- Supported Image Type :
  pix,cut,dcx,gif,jpg,pcd,bmp,ico,cur,mng,psp,psd,pic,pbm,pgm,pnm,ppm,png,sgi,bw,rgb,rgba,tga,vda,icb,vst,tif,xpm,pc
- You can also open an image from web site directly





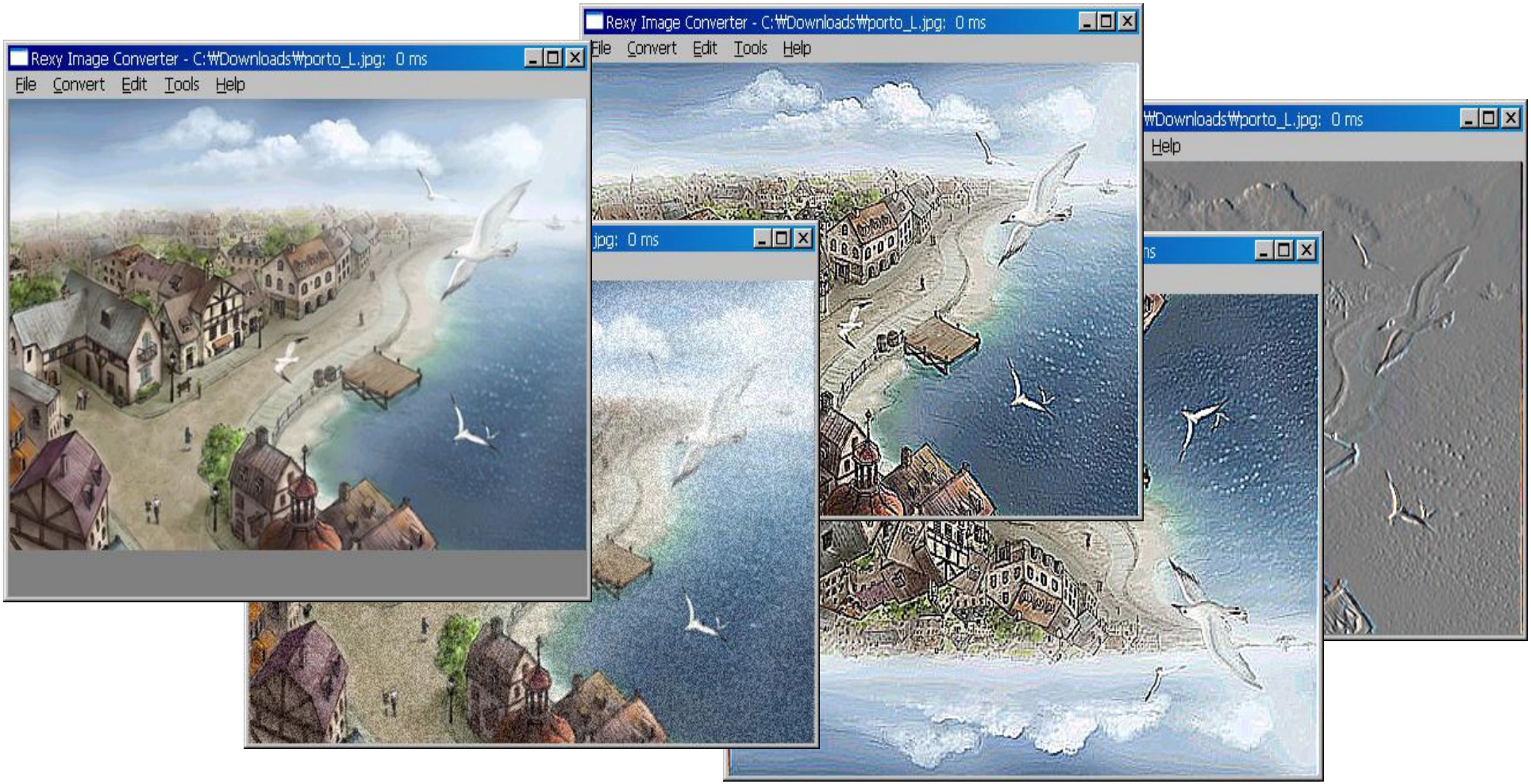
- Supports command line execution so that you can batch your routine job !

```
C:₩Temp₩ResConvTest>imgConverter -FA bmp13by13.bmp outputName varName

C:₩Temp₩ResConvTest>imgConverter bmp13by13.bmp outputName2 varName2

C:₩Temp₩ResConvTest>_
```

Also it provides useful filters.



2. TECHNICAL HIGHLIGHTS

2.7 Comparing with other GUI libraries

	Rexy Embedded™	Zinc	microGUI	picoGUI	Qtopia
operating system	RTOS, Rex, µC/OS-II, iTron, Embedded Linux, Windows	VxWorks, DOS	QNX real time platform	Linux	Embedded Linux
H/W	ARM, x86, DragonBall, Hitachi SH, Various Embedded System CPU	x86, PowerPC, MIPS, SH4, StrongARM	ARM, MIPS, SH4, PowerPC, Xscale, x86	X86	x86, ARM, StrongARM frame buffer driver
API looks like	Win32-like	C++	It's own API	C	C++
easy to port (HAL)	YES	NO (OS dependent)	YES	NO (OS dependent)	NO (OS dependent)
executable binary size	40 KB	350 ~ 750 KB	Under 1 MB	A few hundred KB	Phone & PDA Edittion - Flash : 8~16MB - SDRAM :16~32MB
double byte code system	Support	Support	No	No	Support
resolution	1, 2, 8, 12, 15, 16, 24 bit plain LCD, with no size restriction	unknown	24 bit only (Dithering may be needed)	Default: 1024 x 768, 32 bit	Phone Edition : 176x208, 24bit color PDA Edition : (240x320, 480x640) Core : Various screen Size
Included tool kit	simulator, font converter Image converter	Resource editor	QNX neutrino - PhAB	none	Linguist, Qmake, Designer(Rsrc Editor)

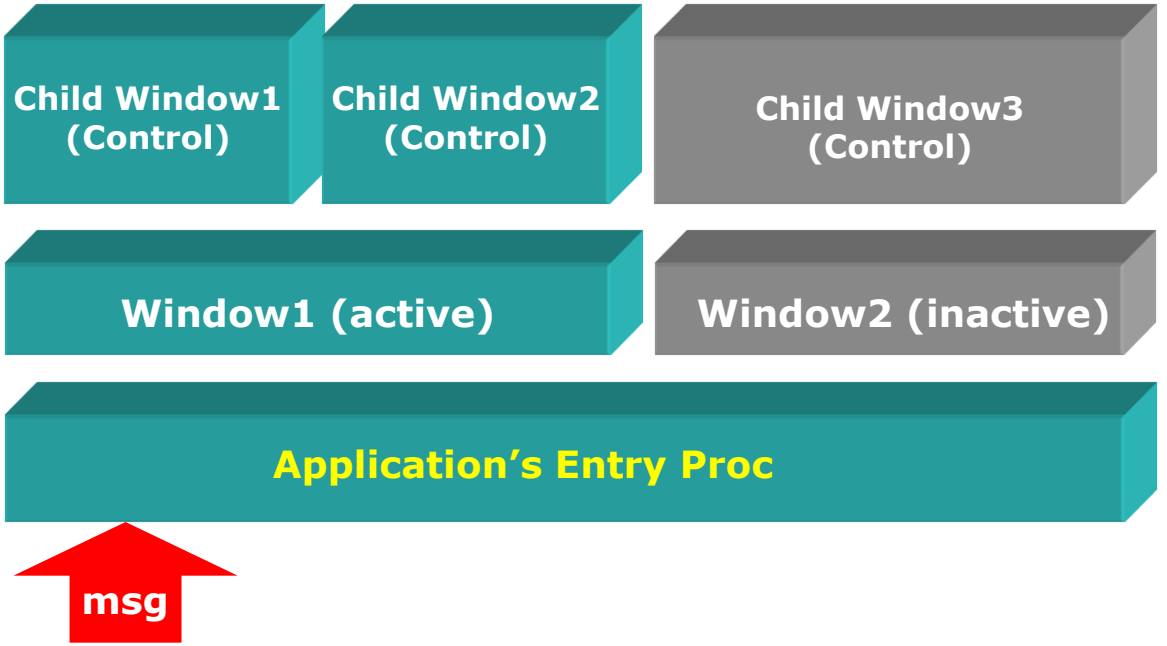
3. APPLICATION DEVELOPMENT

1. Overview
2. Applications
3. How to Use Controls

Making Application !

- Application Entry Procedure

```
int HelloWorldProc(WND hWnd, int nMsg, int
wParam, int lParam)
{
   int bHandled = FALSE;
   switch (nMsg)
   {
      case AWM_CREATE:
      // Create your window here
      bHandled = TRUE;
      break;
      case AWM_DESTROY:
      // Do something to destroy your application
      bHandled = TRUE;
      break;
   }
   // Return 'TRUE' if you consume this message
   return bHandled;
}
```



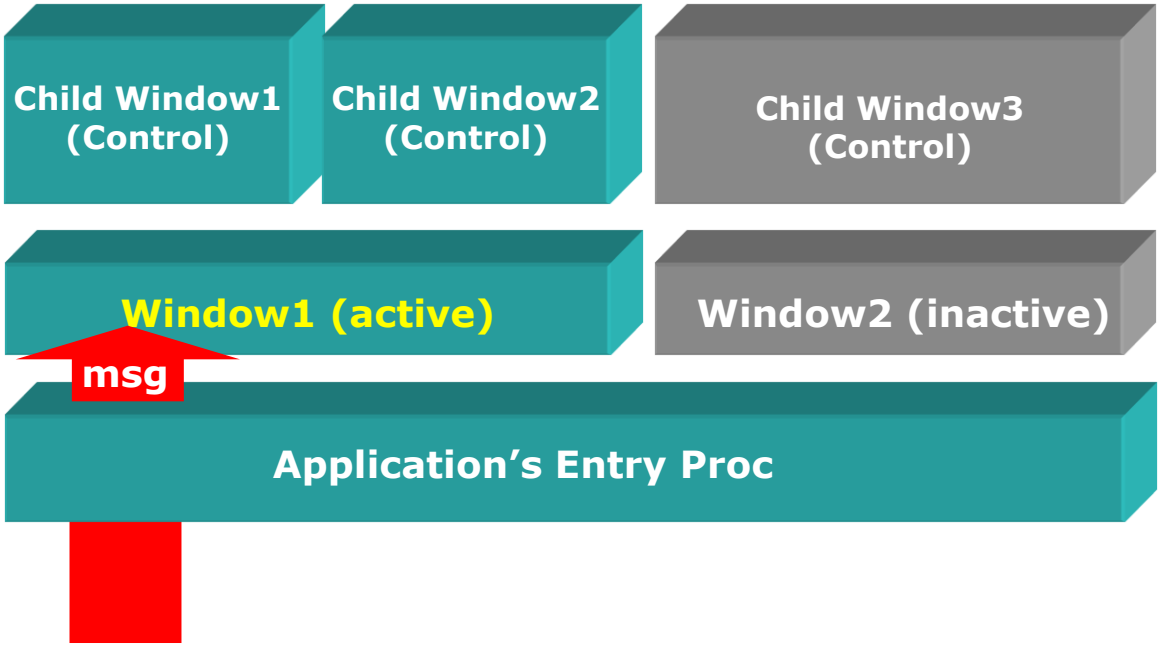
An application may have one or more windows, and an window may have one or more child windows

http://nemustech.com Yuhwa Bldg. 2F, 708-34 Yeoksam-Dong Gangnam-Gu, Seoul, 135-919, Korea

If the application's entry proc did not consume the message…

- Window Message Procedure

```
int Window1MsgProc(WND hWnd, int nMsg, int
wParam, int lParam)
{
   int bHandled = FALSE;
   switch (nMsg)
  {
      case AWM_WINCREATE:
      // Initialize structures, if needed
      bHandled = TRUE;
      break;
      case AWM_WINDESTROY:
      // Do something to destroy your window
      bHandled = TRUE;
      break;
  }
  // Return 'TRUE' if you consume this message
   return bHandled;
}
```

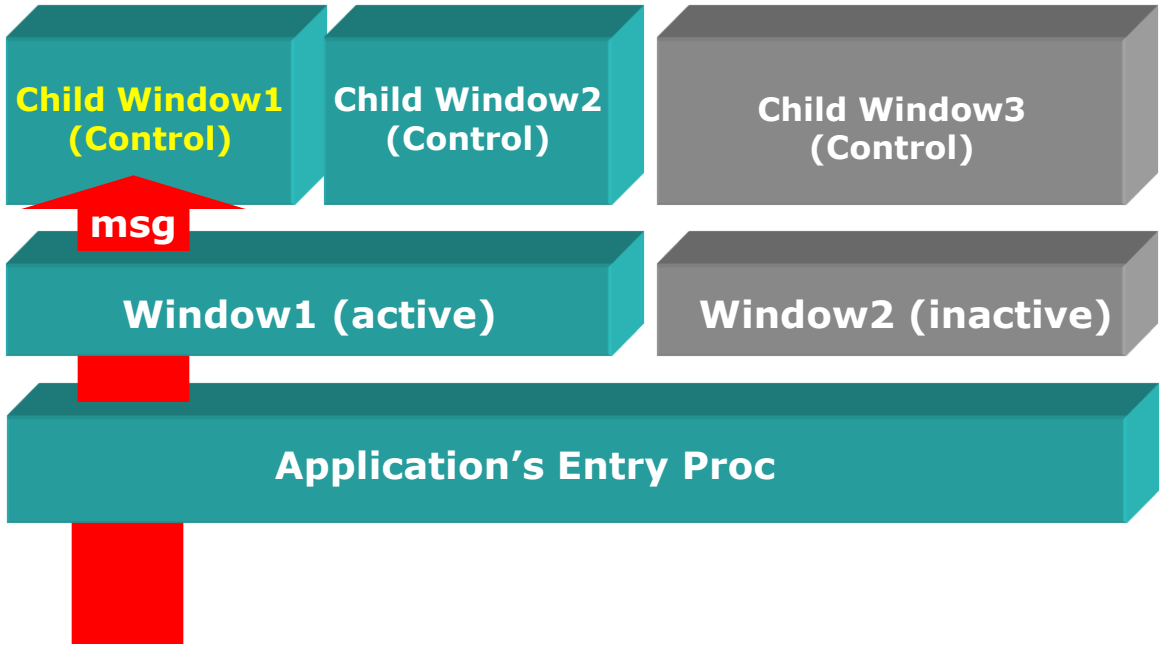


An application may have one or more windows, and an window may have one or more child windows

If the window1's message proc did not consume the message…

- Child Window1 / Control Message Procedure

```
int Child1MsgProc(WND hWnd, int nMsg, int
wParam, int lParam)
{
   int bHandled = FALSE;
   switch (nMsg)
  {
      case AWM_WINCREATE:
      // Initialize structures, if needed
      bHandled = TRUE;
      break;
      case AWM_WINDESTROY:
      // Do something to destroy your window
      bHandled = TRUE;
      break;
  }
  // Return 'TRUE' if you consume this message
   return bHandled;
}
```

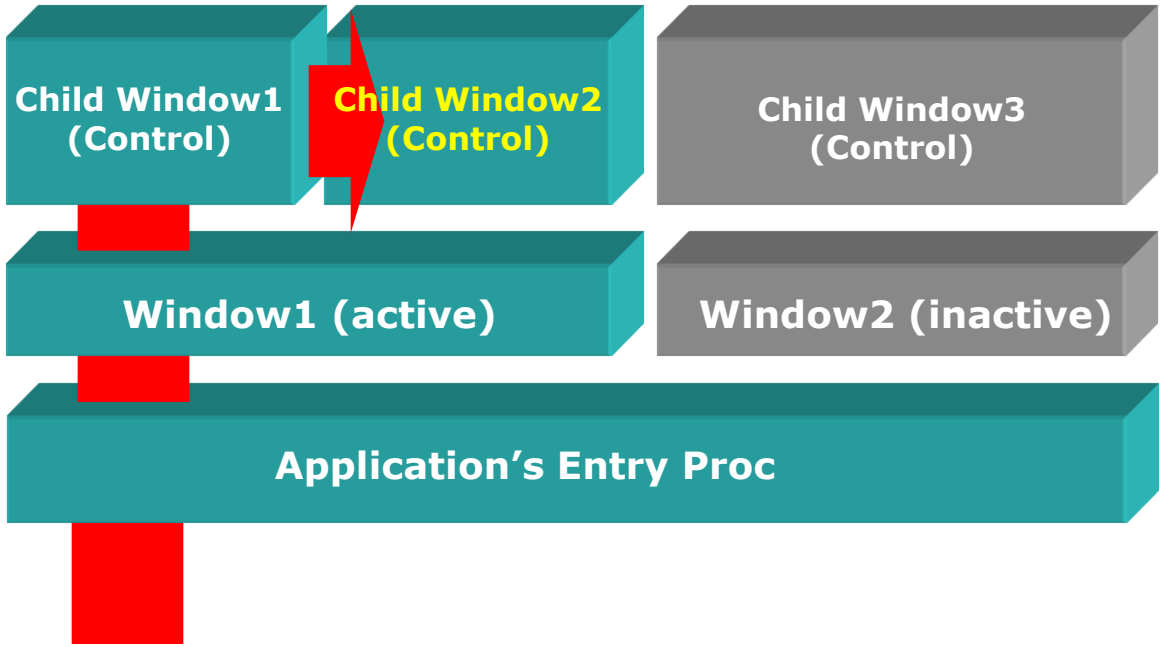


Controls are sort of windows

If the Child Window1's message proc did not consume the message…

- Child Window2 / Control Message Procedure

```
int Child2MsgProc(WND hWnd, int nMsg, int
wParam, int lParam)
{
   int bHandled = FALSE;
   switch (nMsg)
  {
      case AWM_WINCREATE:
      // Initialize structures, if needed
      bHandled = TRUE;
      break;
      case AWM_WINDESTROY:
      // Do something to destroy your window
      bHandled = TRUE;
      break;
  }
  // Return 'TRUE' if you consume this message
   return bHandled;
}
```

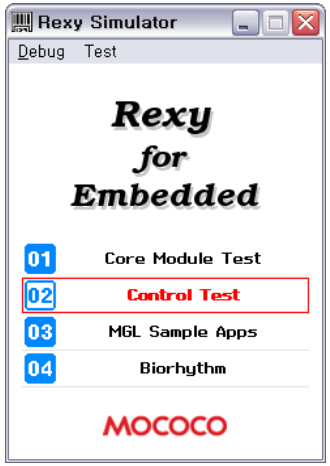


Controls are sort of windows

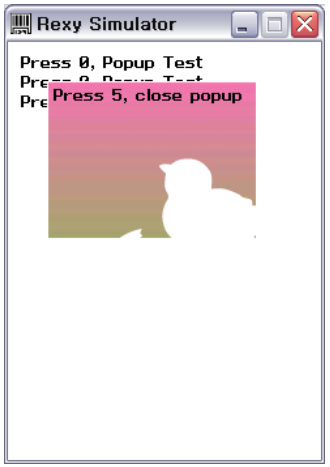
Rexy Embedded™ provides various control sets and more !

- Commonly used controls are already built in Rexy Embedded™ !
- You can customize built-in controls and create new controls using provieded control-core module.

List

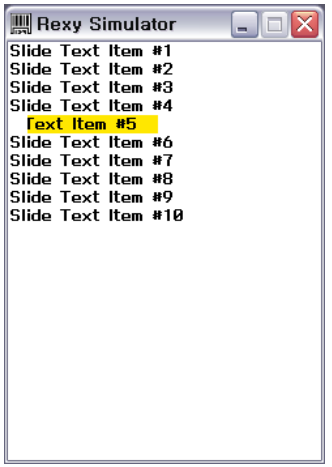


Popup



Edit

Slide Text



Animation Bitmap

Message

- A logical concept that informs event to an application or its subsidiary windows.
- Message ID and two integer values (lParam, wParam).

Type of Messages

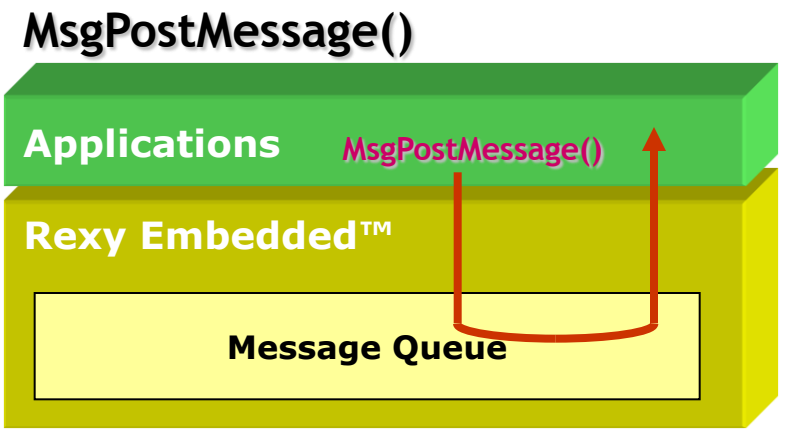
- AWM_CREATE
- AWM_DESTROY
- AWM_LOAD
- AWM_SAVE
- AWM_WINCREATE
- AWM_WINDESTROY
- AWM_PAINT
- AWM_TIMER
- AWM_QUIT
- AWM_RETURN
- AWM_CLOSE_POPUP
- AWM_KEYDOWN
- AWM_KEYUP
- AWM_KEYREPEAT
- AWM_SETFOCUS
- AWM_KILLFOCUS
- Etc …

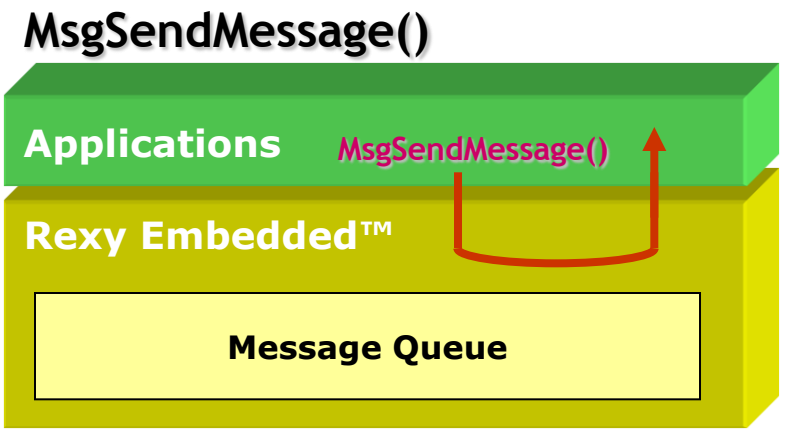
Message Delivery Order

- Rexy Embedded™ gets event messages from the focused window.
- At first, messages are delivered to application's entry state.
- If entry state doesn't treat the messages, they will be sent to the top main window.
- Messages will be passed over parent window to child controls until any window treats them.
- If an event message is a timer message, Rexy Embedded™ will send it to target window directly.

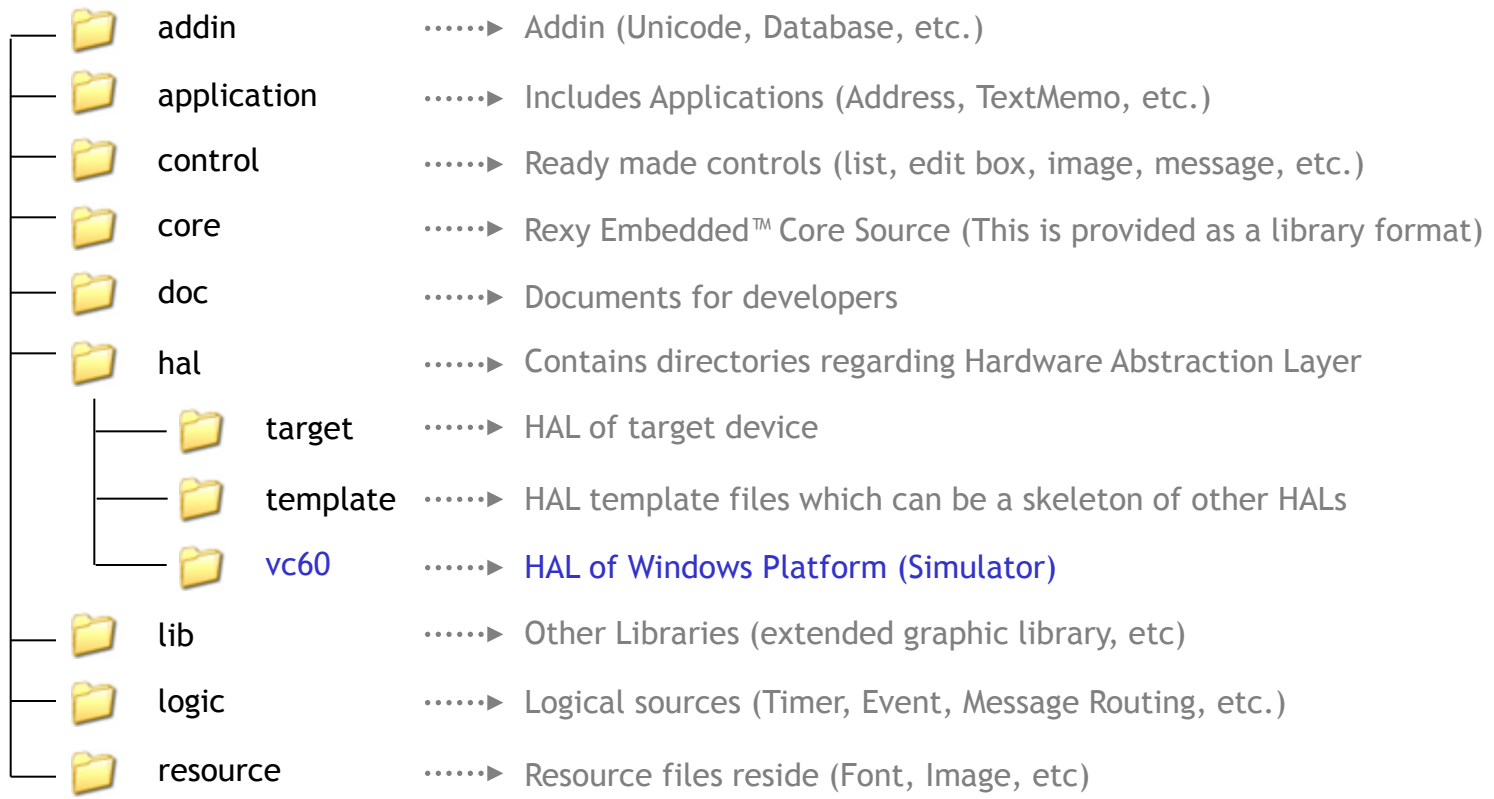
Message Delivery Method

- Message delivery methods are classified according to the usage of message queue.

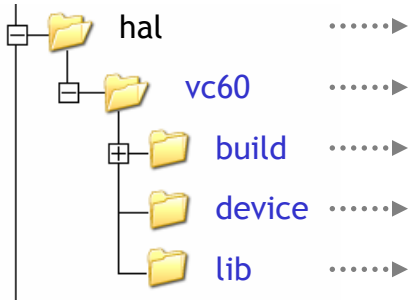




- This directory structure can vary according to your designated device



Simulator Folders



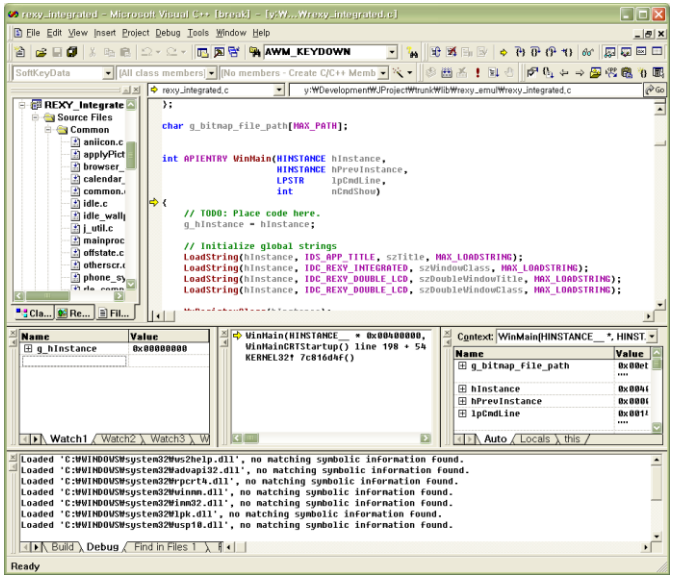
- hal ····▶ Contains directories regarding Hardware Abstraction Layer
  - vc60 ····▶ HAL of Windows Platform (Simulator)
    - build ····▶ Project files of which you can find work space file for Visual C++ reside
    - device ····▶ Device dependent layer files
    - lib ····▶ Compiled result will be placed as a library format

Build Environment

- Windows95 or higher
- Visual Studio / Visual C++
- Pentium3 CPU 500MHz or higher is recommended

Build Process

- Start MS Visual C++ with work space file (.dsw) in build directory
- Set Emulator project as active project
- Build project and Execute !
- Test & debug program on Simulator with Visual C++'s assistance



Procedure of making an application

Generally an application consists of one state and the state may have several windows.

- Define the STATE_ID and message handler for the application's entry and map them.
- Create a window If you needs to display something on the screen.
- Make your own codes for the events in the specified window's message handler.
- Destroy the window when you finish your job.

- Let's assume 'STATE_ID_HELLO_WORLD' as STATE_ID and define in './application/application.h' file.

```
typedef enum {
    STATE_ID_ENTRY = 0,
    STATE_ID_HELLO_WORLD,
    // ...
} AppStateIDType;
```

- Then Mapping of STATE_ID and message handler for STATE_ID is defined in AppRegister() in "./application/application.c" file. Assume the 'HelloWorldProc()' message handler for 'STATE_ID_HELLO_WORLD'.

```
static void AppRegister()
{
    StaSetEventHandler( STATE_ID_ENTRY,       EntryProc );
    StaSetEventHandler( STATE_ID_HELLO_WORLD, HelloWorldProc );
}
```

'STATE_ID_ENTRY' is special STATE_ID and it is mapped to message handler for the entry point.

- Also 'HelloWorldProc()' message handler should be defined in './application/HelloWorld/HelloWorld.c' file.
- To draw something on screen, application must create the window.
- Window should have a message handler and it can be mapped by using WinCreateWindow() function.
- When your job finished, window must be destroyed using WinDestroyWindow().

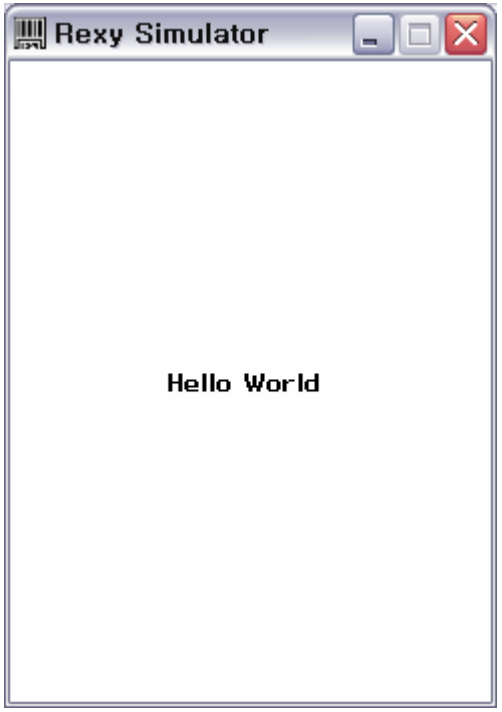
```
int HelloWorldProc(WND hWnd, int nMsg, int wParam, int lParam)
{
    int bHandled = FALSE;
    static WND hHWWnd;

    switch (nMsg) {
    case AWM_CREATE:
        hHWWnd = WinCreateWindow( HelloWorldWinProc, x, y, width, height, WND_PANEL );
        WinUpdateWindow ( hHWWnd, 0 );
        bHandled = TRUE;
        break;
    case AWM_DESTROY:
        WinDestroyWindow( hHWWnd );
        bHandled = TRUE;
        break;
    }
    return bHandled;
}
```

- To draw “Hello World” on the screen, application may use GDI API.

```
int HelloWorldWinProc(WND hWnd, int nMsg, int wParam, int lParam)
{
    int bHandled = FALSE;
    char *szText = “Hello World”;

    switch (nMsg) {
    case AWM_PAINT:
        GdiTextOut( hWnd, szText, StrLen( szText), x, y);
        bHandled = TRUE;
        break;
    }
    return bHandled;
}
```



How to use controls

[Essentiality]

1. Create a control.
2. Set initial properties of control.
3. Manipulate it as you want (ex: get/set properties of the control).
4. Destroy the control.

[Addition] – For user-defined behaviors

1. Implement a callback message handler and register it into the control when you create it.

- The following codes will create a sliding text control which can be toggled through pressing a key.

```
int HelloWorldWinProc(WND hWnd, int nMsg, int wParam, int lParam)
{
    static WND hCtlWnd = NULL;
    int bHandled = FALSE;
    int nErr;
    int bHandled = FALSE;
    const rect rtClient = {0, 160, 240, 180};

    switch (nMsg) {
    case AWM_WINCREATE:
        hCtlWnd = CtlCreate( hWnd, CTL_TYPE_SLIDETEXT, NULL, NULL,&nErr);
        CtlSetPtr( hCtlWnd, CTL_SLIDETEXT_CMD_SET_WINRECT, (void *)&rtClient, &nErr);
        CtlSeti( hCtlWnd, CTL_SLIDETEXT_CMD_SET_TIMEOUT, 100, &nErr);
        CtlSeti( hCtlWnd, CTL_SLIDETEXT_CMD_SET_MAXWIDTH, 50, &nErr);
        CtlSetPtr( hCtlWnd, CTL_SLIDETEXT_CMD_SET_DATA, (void *)"Hello Control", &nErr);
        CtlSetPrt( hCtlWnd, CTL_SLIDETEXT_CMD_SET_CB_DRAW, (void *)HelloWorldDrawCB, &nErr);
        CtlDo( hCtlWnd, CTL_SLIDETEXT_CMD_DO_DRAW, 0, 0, &nErr);
        bHandled = TRUE;
        break;

    case AWM_WINDESTROY:
        CtlDestroy( hCtlWnd, &err );
        hCtlWnd = NULL;
        hHandled = TRUE;
        break;
```

Continue.

Continued.

```
    case AWM_KEYDOWN:
        if (bPlayed)
            CtlDo( hCtlWnd, CTL_SLIDETEXT_CMD_DO_STOP, 0, 0, &nErr);
        else
            CtlDo( hCtlWnd, CTL_SLIDETEXT_CMD_DO_PLAY, 0, 0, &nErr);
        bPlayed != bPlayed;
        bHandled = TRUE;
        break;
    }
    return bHandled;
}
```

- Before drawing, control callback function is called. This sample code will change the foreground and background color of the control.

```
int HelloWorldDrawCB(WND hCtlWnd, COLORVAL nFgColor, COLORVAL nBgColor )
{
    int nErr;
    rect rtClient;

    if (nFgColor != COLOR_RED)
    {
        nFgColor = COLOR_RED;
        nBgColor = COLOR_BLUE;
    }
    else
    {
        nFgColor = COLOR_BLUE;
        nBgColor = COLOR_RED;
    }

    CtlSeti (hCtlWnd, CTL_SLIDETEXT_CMD_SET_FGCOLOR, nFgColor, &nErr);
    CtlSeti (hCtlWnd, CTL_SLIDETEXT_CMD_SET_BGCOLOR, nBgColor, &nErr);
    GdiSetRect (&rtClient, 0, 0, 240, 20);
    GdiFillRect( hCtlWnd, &rtClient, COLOR_BLUE);

    return 0;
}
```

4. CONTROL DEVELOPMENT

1. Overview
2. Controls

Control related message handler and callback functions

Controls have 3 control handler, 1 application callback message handler and optional several control's callback functions.

- **Control Handler**
  - Main message handler
  - Callback message handler
  - Command handler

- **Application callback message handler**

  This handler is used to customize message processing in an application.

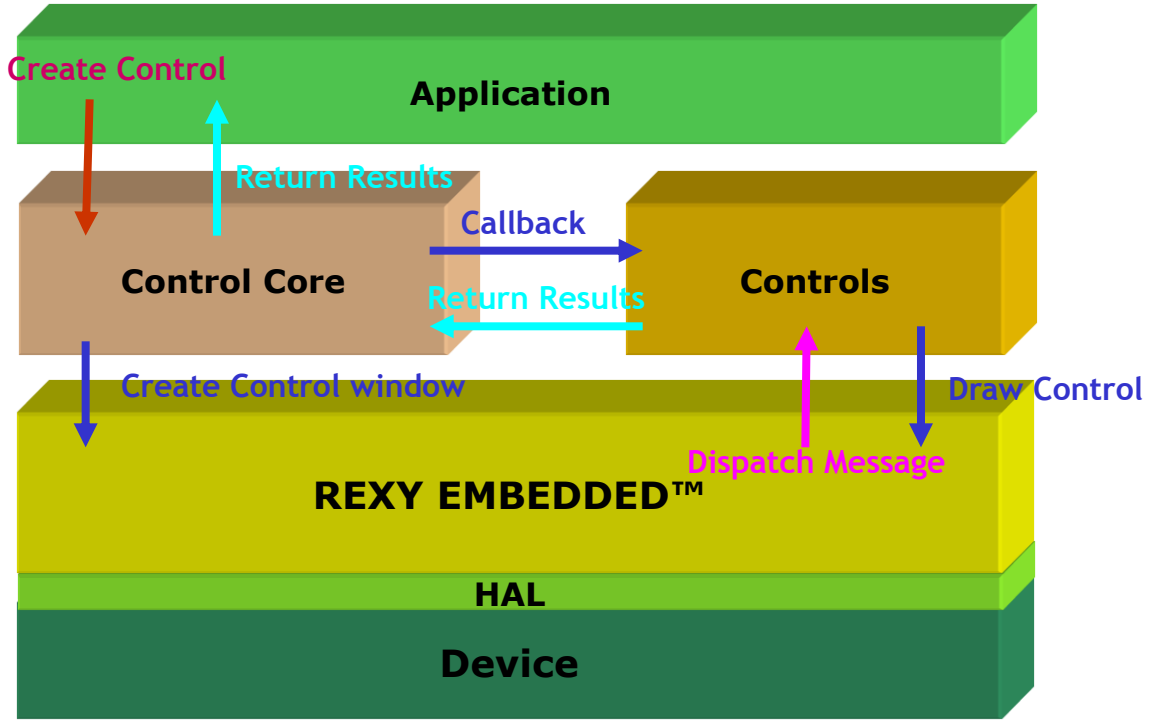
- **Control's Callback Function**

  This callback function is used to customize the behaviour of control.

How controls work

- All control is the child window of application window.
- Once an application creates a control, it has it's own message handler.
- Control core calls callback function that describes real control.
- Whole messages will be dispatched to control window.

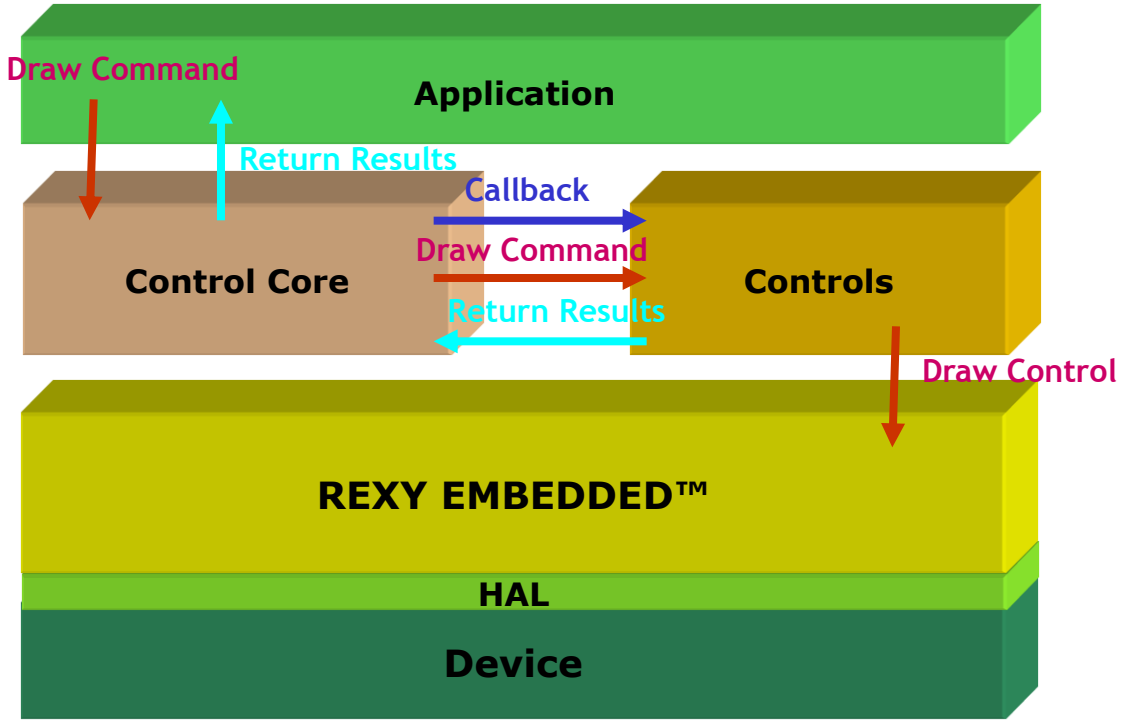
- Control Creation Flow Order
  1. Application creates a control
  2. Create a control window that has its message handler and draw it
  3. Dispatch events to control window
  4. Control handles messages and returns results to its parent application



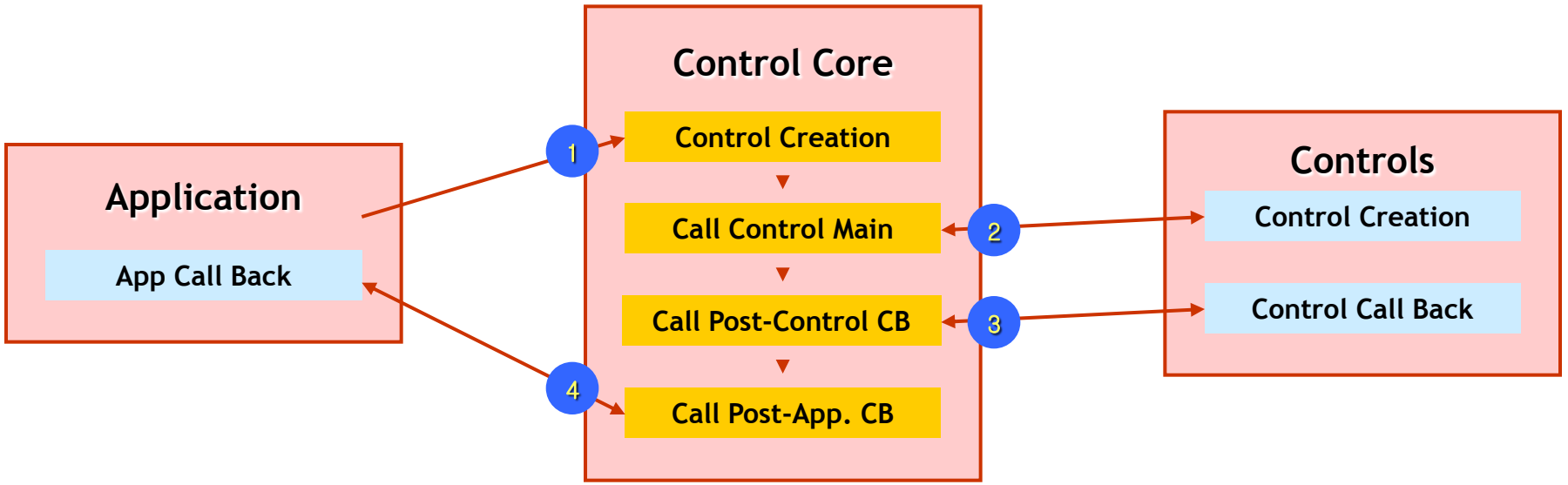
Control Command

- Application can order command to control for various actions.
- Most of control manipulations are executed by commands.

- Command Flow Order(draw)
  1. Application orders command to control
  2. Control core calls callback and control command handler to execute command
  3. Control returns results to its parent application

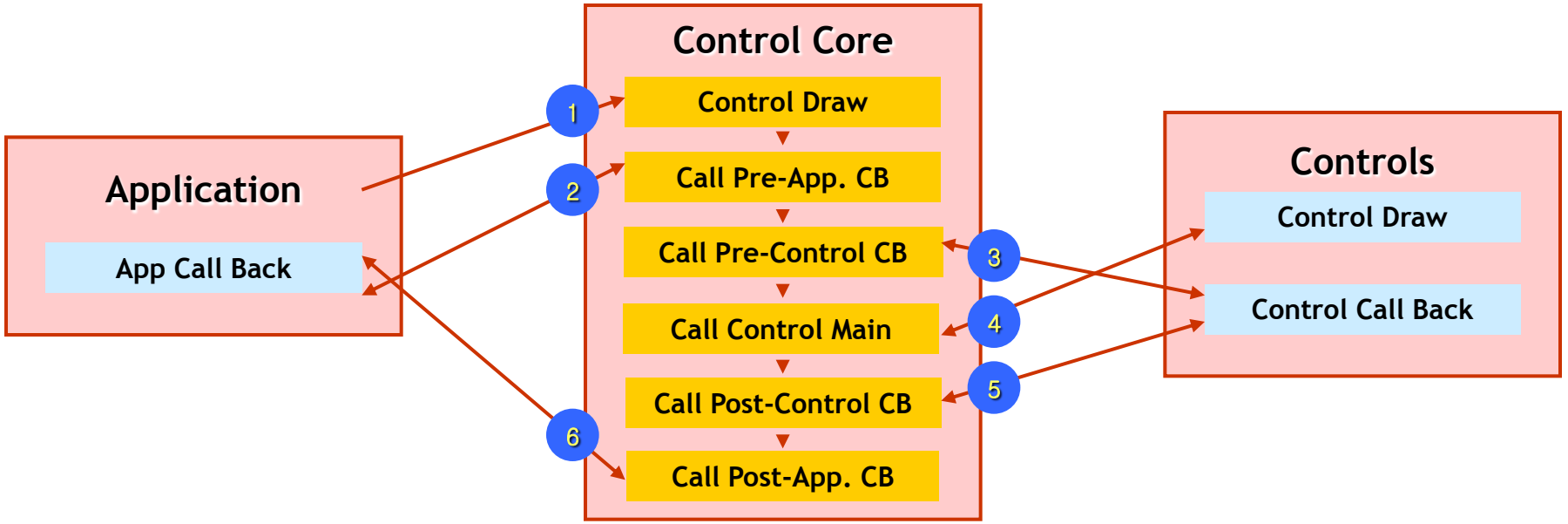


Creation Flow



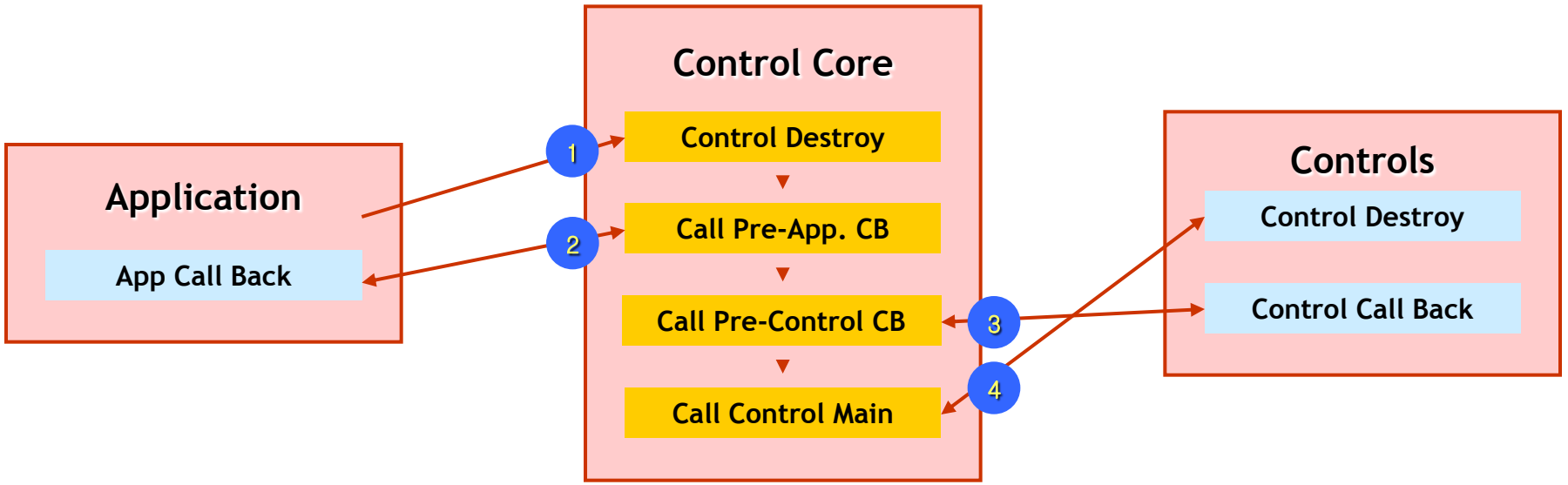
1	Application calls control creation function
2	Control Core creates new control structure
3	Control Core calls proper control creation function (Ex. If application creates list control, Control Core calls the creation function of the list control)
4	Finally, Control Core calls application callback function and application handles create process

General Flow (Draw / Load / Save Command)



1	Application calls control Draw/Load/Save function
2	Control Core calls pre-handle application callback function about the each Draw/Load/Save message
3	Control Core calls pre-handle control callback function about the each Draw/Load/Save message
4	Control Core calls control main callback function about the each Draw/Load/Save message
5	Control Core calls post-handle control callback function about the each Draw/Load/Save message
6	Control Core calls post-handle application callback function about the each Draw/Load/Save message

Destroy Flow

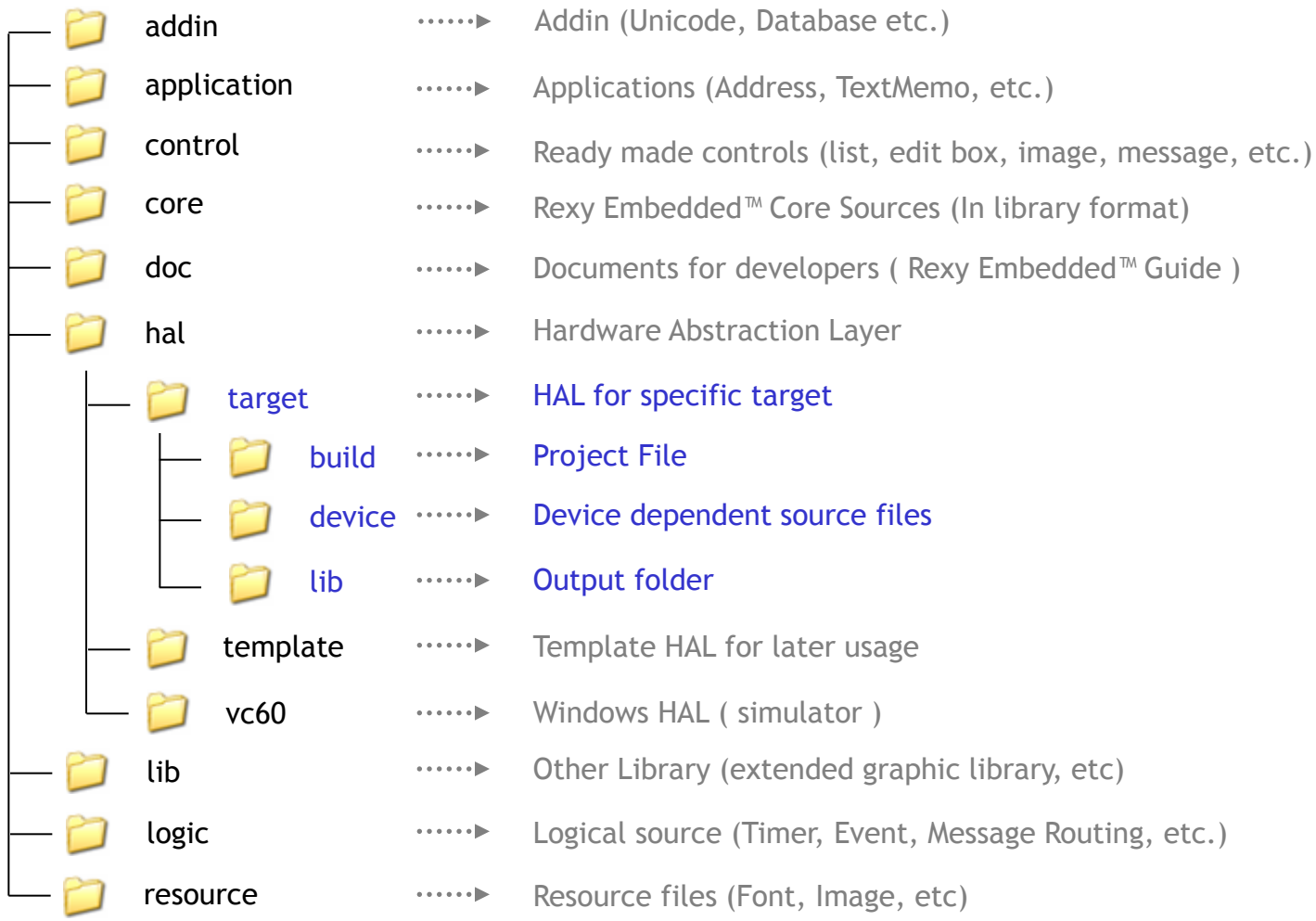


1	Application calls control destroy function
2	Control Core calls application callback function so that application handles destroy process
3	Control Core calls proper control destroy function (Ex. If application destroy list control, Control Core calls the destroy function of the list control)
4	Finally, Control Core destroy control structure

5. Porting into Target

1. Overview
2. Case Study

- This directory structure can vary according to your designated device kind



- Build environment
  - Windows 95 or higher
  - Building Tools provided by device chipset provider (ex: ADS, HEW)
  - Download program
  - Cable (Serial / USB)
  - Debugging tool (ex: Trace32, E10A..)

- Build Process
  - Start Device dependent IDE with its work space file (ex: .hws) in build directory
  - Build executable binary file using device cpu's compiler
  - Download using cable to device
  - Test & debug using device's debugging tool (This depends on the device provider)

Porting Issues

- Message Loop
- Key & Other Event
- Timer
- Display
  - Resolution
  - Color depth
  - Number of Screen
- Standard C Library
- External Resource (Optional)

Case Study

- MX-400 Media Player

Category	Item	Details
AUDIO	Supported Type	MPEG1/2/2.5 Layer3, WMA, ASF, OGG
VIDEO	Supported Type	AVI, MPEG1, MPEG2, MPEG4, WMV
Display	Color LCD	260,000 color, 1.3” TFT LCD (160 x 128)
Power	Internal Battery	Li-Polymer, Max 20 hours continuous playback capacity
Features	USB	Support ver 2.0
	Lyric	Automatic Lyric (LDB)
	Text Viewer	Support Multi-Language
	Image Viewer	Support 260,000 color image preview
	Remote control	TV
	Game	Puzzle, Stopwatch
	Recording	Voice, FM Radio, Line-in
	Language	Korean,English,Japanese,Chinese,Russian,German,French

Category	Item	Details
System	PC	Windows 98/SE/ME/2000/XP, Mac9.0, Linux 2.4
Hardware	Size	74.0 mm(L) x 33.6mm(W) x 19.5 mm(H)
Display	Weight	Approx. 48.5g (including battery)
Processor		MCSLogic MLC3590

5. PORTING INTO TARGET

2. Case Study – MLC3590 CPU

- System On Chip
  - System Controler: ARM946ES core (32bit RISC Processor)
  - Decoder for multi-format digital audio players
    - MP3, WMA, OGG

- Audio Functions
  - MPEG1/2/2.5 layer 2 and 3 decoding
  - Window Media Audio (WMA V9 compatible) decoding
  - Ogg decoding
  - MPEG1/2/2.5 layer3 encoding
  - Supports Software MUTE / Pause / Resume / volume
  - Digital volume control
  - 7-band sound Equalizer for MP3, WMA, Ogg and Red book audio CD
  - 7 band graphic Equalizer for MP3, WMA, Ogg and Red book audio CD
  - Optional sampling rate conversion to 44.1Khz for off-chip general audio DAC
  - Channel mixing or separating for two different audio sources
  - Supports time display (Normal / FF / FB)
  - MEBB (MCS Logic Enhanced Bass band) sound algorithm
  - CD-TEXT decoding in Lead-in area
  - High quality ESP sound with high compression rate

- USB1.1 Host controller
  - USB Rev1.1 compatible
  - Open HCI Rev1.0 compatible
  - Support for both Low Speed and Full Speed USB Devices

- USB2.0 Device controller
  - Compliant to USB2.0 specification
  - Support HS(High Speed, 480Mbps) transfers, as well as USB1.1 FS (Full Speed, 12Mbps)
  - Support Interrupt, Bulk Transfer

- NAND Flash/SMC interface
  - Support 8bit/16bit data interface.
  - Two independent chip select signal

- IDE / ATA interface
  - Supports ATA PIO Modes 0-4 UDNA Mode Ø-4
  - Supports CF card interface

- CD Format Decoding Functions
  - ID3 tag V1.1 and V2.3 extraction
  - Supports ISO9660 CDROM Mode1/Mode2 format (CDROM-XA)
  - Supports Joliet decoding both single-session and multi-session disc
  - Supports Joliet Level 3 decoding
  - Supports UDF V1.02/V1.5/V2.01
  - Supports UDF decoding in Packet writing format on CD-RW disc
  - Supports UDF decoding in single/multi session Non-packet

MLC3590

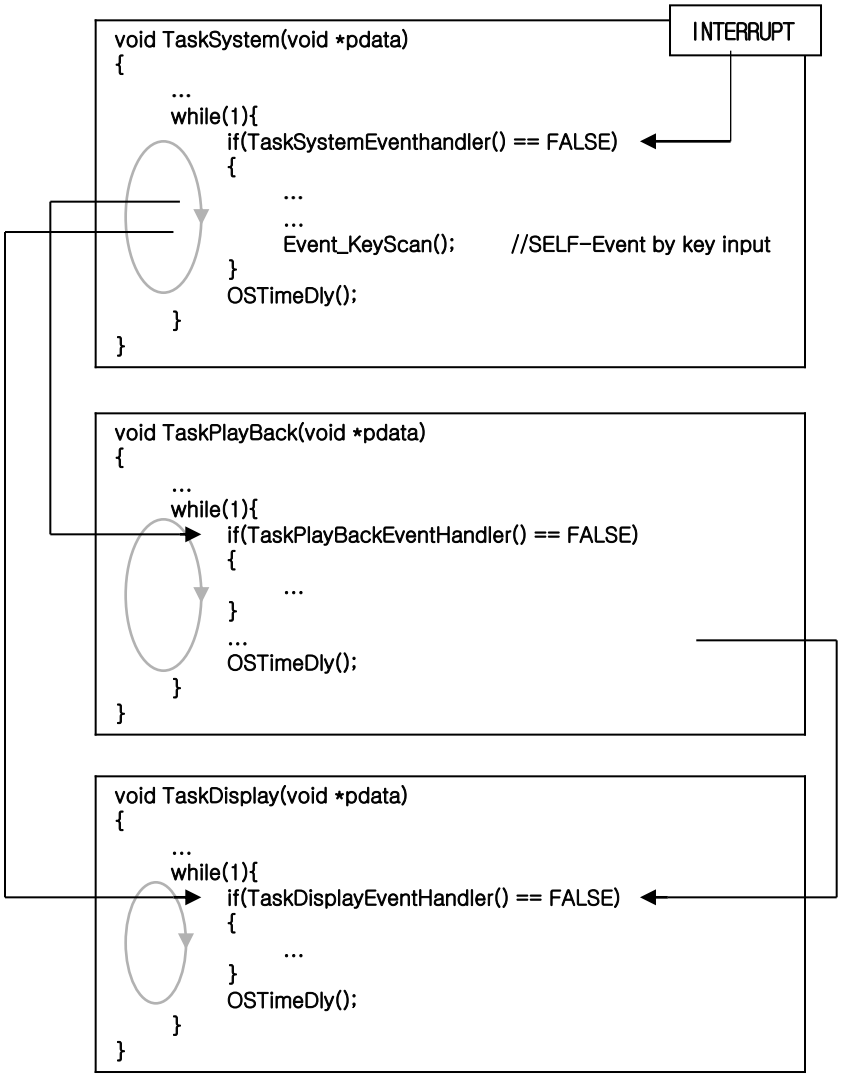
High-Performance Digital Audio Processor

Programming Model

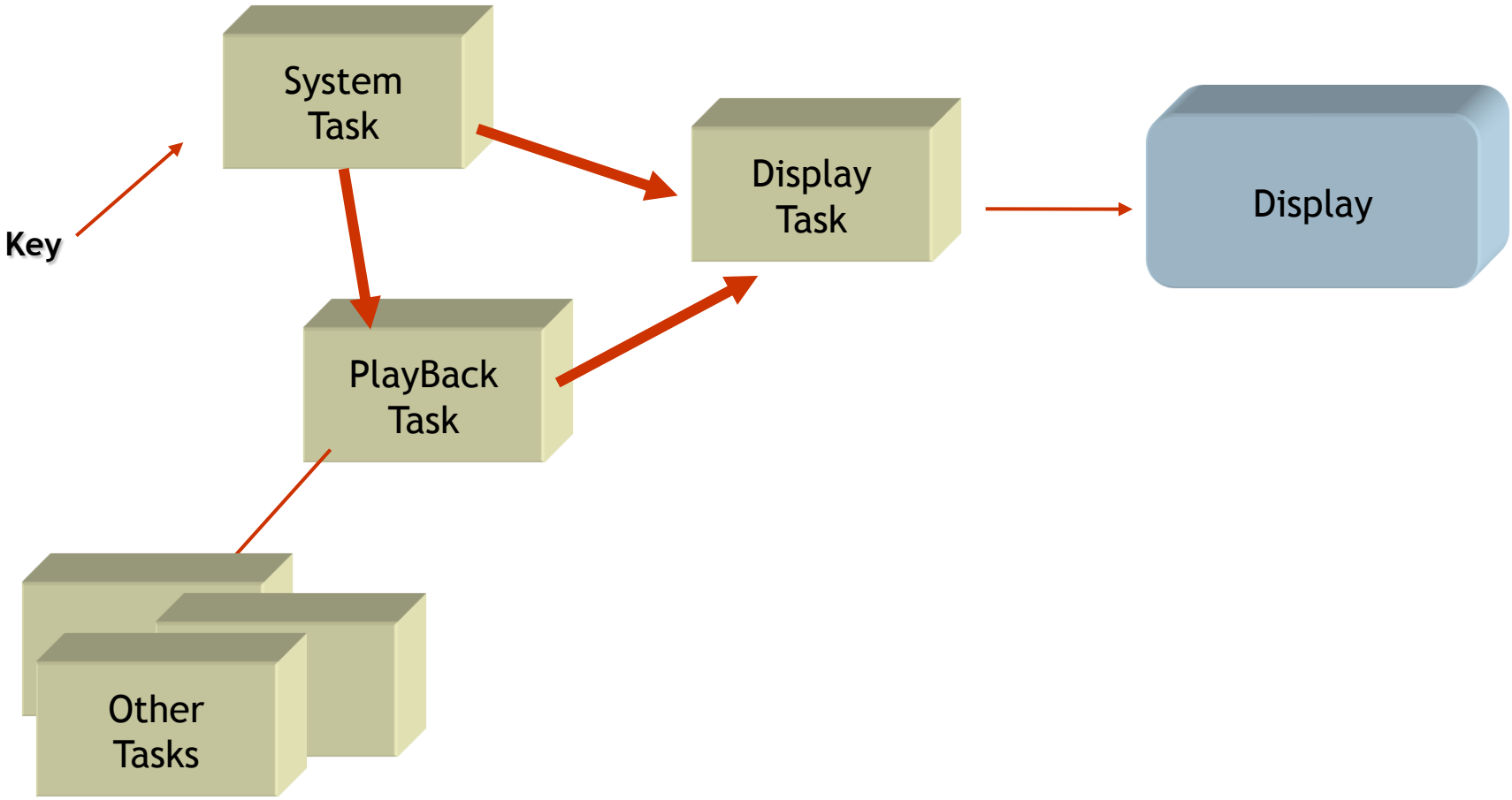
- State-Based
- Event-Driven

Tasks

- USB Device/Host task
- TaskBuffering
- TaskAVI
- TaskAudio
- ***TaskSystem***
- ***TaskDisplay***
- TaksPlayBack
- TaskBufferingExt
- TaskOpenDisk



Event Flow on MLC SDK



Event Handling Code in MLC SDK

- TaskSystem.c:TaskSystemEventHandler()

```
U32 TaskSystemEventHandler(void)
{
        void *ESG_MSG;

        if((ESG_MSG = Event_Receive(TaskSystemQ)) != (void *)0){
                Event_Handler(gpasTaskSystemStates[gucTSystemState.iCurrState],(sEvent_MSG*)ESG_MSG,&gucTSystemState);
                return TRUE;
        }
        return FALSE;
}
```

```
Event_Handler(
                gpasTaskSystemStates[gucTSystemState.iCurrState],       // Event Handler in Current State
                (sEvent_MSG*)ESG_MSG,                                   // Event
                &gucTSystemState);                                      // State
```

Event Handling Code in MLC SDK

- eTaskSystem.h:gpasTaskSystemStates

```
const sStateType * const gpasTaskSystemStates[] =
{
                gasTSystemInit,                 // TSYSTEM_INIT_STATE
                gasTSystemNavi,                 // TSYSTEM_NAVI_STATE
                gasTSystemMenu,                 // TSYSTEM_MENU_STATE
                gasTSystemPlay,                 // TSYSTEM_PLAY_STATE
                gasTSystemRec,                  // TSYSTEM_RECORD_STATE
                gasTSystemStop,                 // TSYSTEM_STOP_STATE
                gasTSystemDiskChange,           // TSYSTEM_DISKCHANGE_STATE
                gasTSystemFM,                   // TSYSTEM_FM_STATE
                gasTSystemUSB,                  // TSYSTEM_USB_STATE
                gasTSystemPowerDown,            // TSYSTEM_POWERDOWN_STATE
                gasTSystemTime,
#if             defined(__dSYS_SUPPORT_COPY__)
                gasTSystemCopy,
                gasTSystemBackup,
#endif
                gasTSystemEnd
};
```

Event Handling Code in MLC SDK

- eTaskSystem.h:gpasTaskSystemStates

```
const sStateType gasTSystemInit[] =
{
//              Events                  Function                Next State
                {EVENT_KEY_PLAY,        mTSystemStartDec,       TSYSTEM_PLAY_STATE},
                {EVENT_KEY_STOP,        mTSystemStop,           TSYSTEM_STOP_STATE},
                {EVENT_KEY_MENU,        mTSystemNaviInit,       TSYSTEM_NAVI_STATE},
                {EVENT_USB_PLUGIN,      mTUSBInit,              EMPTY_STATE},
                {EVENT_END,             mTSystemNoFunction,     EMPTY_STATE}
};
```

- eTaskSystem.h:mTSystemStartDec()

```
void mTSystemStartDec(sEvent_MSG * psEventMSG)
{
                Event_PlayBackSemPend(EVENT_PLAY_PRESS,NULL);
}
```

Key Event Reading Code in MLC SDK

- TaskSystem.c:TaskSystem()

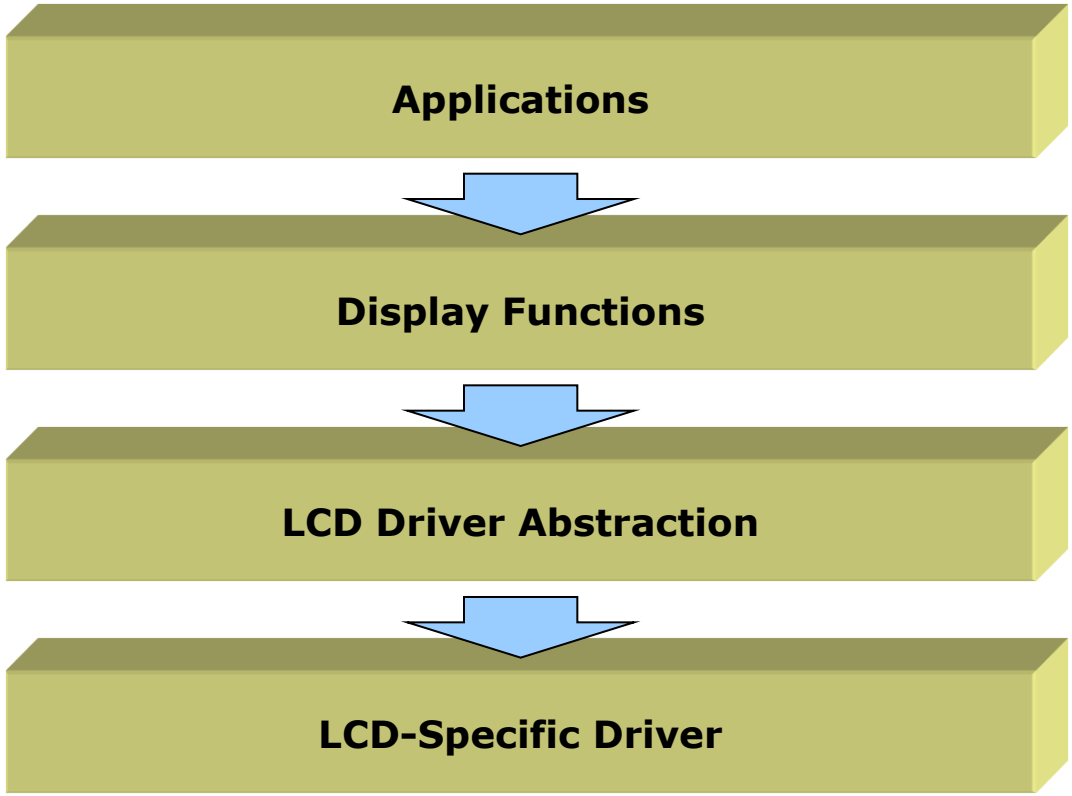
```
void TaskSystem (void *pdata)
{
        // ...
        while( 1 )
        {
                // ...
                if( TaskSystemEventHandler() == FALSE ){
                        // ...
                        Event_KeyScan();
                }
                // ...
        }
}
```

Key Event Sending Code in MLC SDK

- TaskSystem.c:Event_KeyScan()

```
U32 Event_KeyScan(void)
{
KeyScan();
      if( ( iKeyValue != dKEY_NOPRESS ) && ( iKeyValue != dKEY_NOP ) && ( iKeyValue < dKEY_LONG_MAX ) ) {
            //Ignore key input
            // if USB device connection detected
            if( !SystemUSBMonitor() ) {
//                BackLightOn();
                SystemPowerTimeSet();
                if( !iHoldKey ) {
                      //Make TaskSystemQ event up to key value
                      Event_Send(TaskSystemQ, EVENT_NO_EVENT + iKeyValue ,MSG_KEY_TRANSFER);
                      // …
                }
                return TRUE;
            }
            return FALSE;
      }
}
```

Display Code in MLC SDK

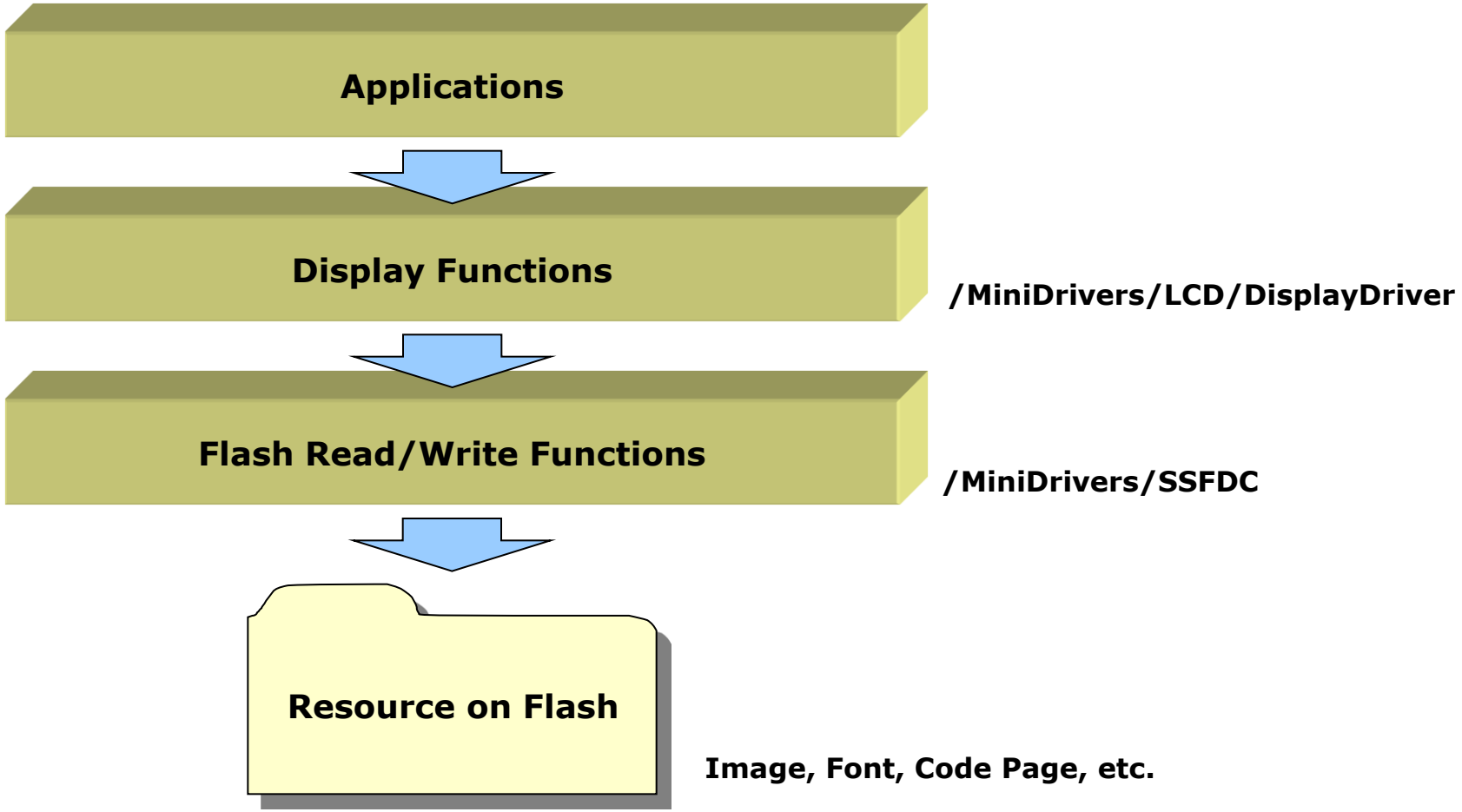


/MiniDrivers/LCD/DisplayDriver

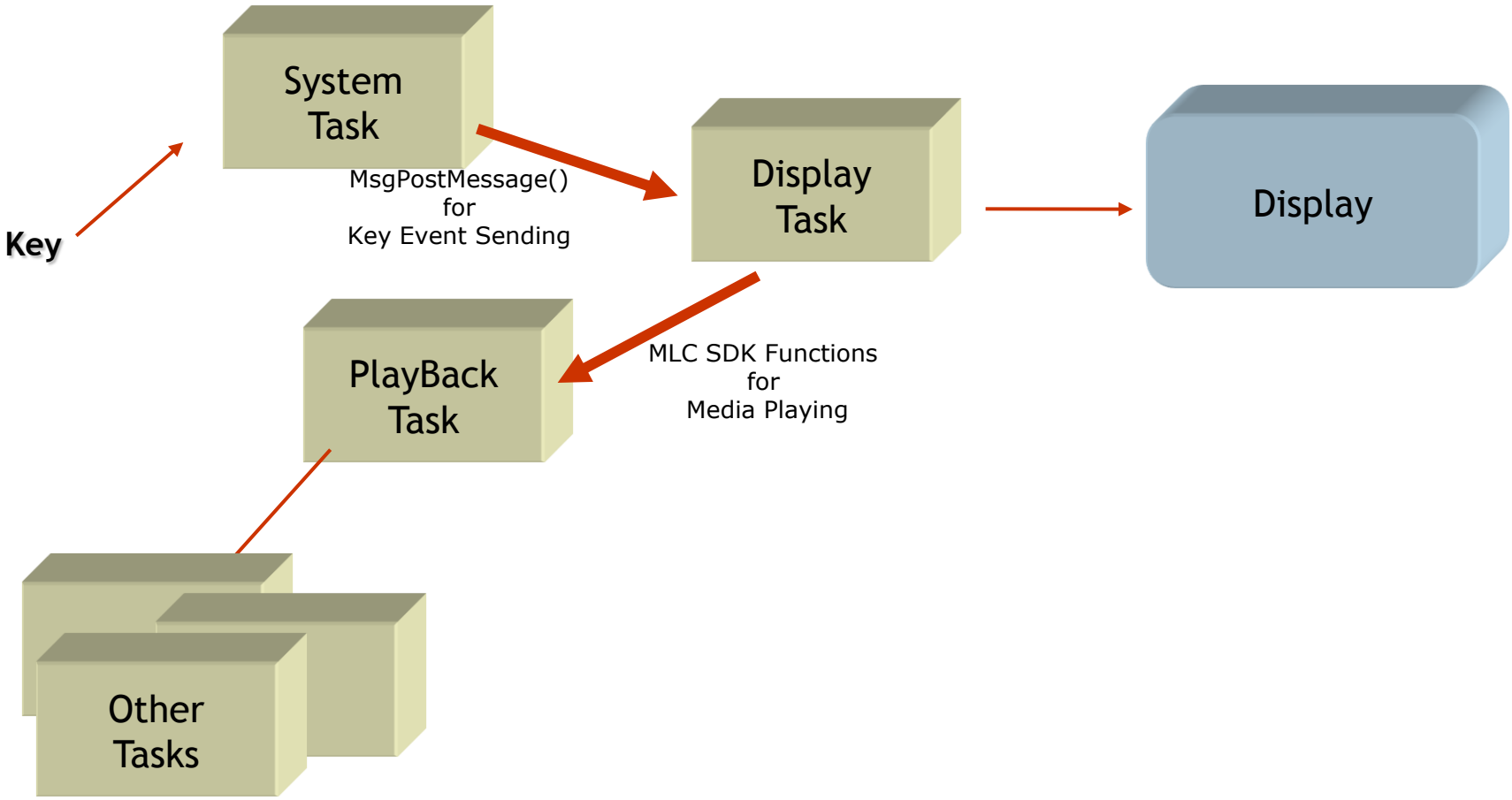
/MiniDrivers/LCD/LCDBasic.h|c

/MiniDrivers/LCD/LCDDriver

External Resources in MLC SDK



Message Loop in Rexy Embedded™



Message Loop in Rexy Embedded™

- TaskDisplay.c:TaskDisplay()

```
void TaskDisplay (void *pdata)
{
   // …
   // Init & Start the RexyE
   StartRexy();

   // …
   while( !ifgSysDeleteActiveTasks ){
      // …
      {  msg nMsg;
         while(MsgGetMessage(NULL, &nMsg)) {
            int ret;
            ret = MsgDispatchSystemMessage(&nMsg);
            if( !ret ) MsgDispatchMessage(&nMsg);
         }
      }
      // …
   }
}
```

Key Message Sending Code in Rexy Embedded™

- TaskSystem.c:Event_KeyScan()

```
U32 Event_KeyScan(void)
{
KeyScan();
      if( ( iKeyValue != dKEY_NOPRESS ) && ( iKeyValue != dKEY_NOP ) && ( iKeyValue < dKEY_LONG_MAX ) ) {
            //Ignore key input
            // if USB device connection detected
            if( !SystemUSBMonitor() ) {
                  if( !iHoldKey ) {
                        //Make TaskSystemQ event up to key value
                        switch (iKeyValue) {
                              // …
                              case dKEY_MENU: MsgPostMessage(WinGetCurrentWindow(), AWM_KEYDOWN, VK_MMENU, 0); break;
                        }
                        // …
                  }
            }
      }
}
```

Timer Code in Rexy Embedded™

- Rexy_halinterface.c

```
static bool          bSetTimer = FALSE;
static ruint32 nTimerTick = 0, nTimerBaseTick = 0;
static int     nTimerId = -1;

bool RalTimeSetTimer(ruint32 inMilliseconds, int id) {
    bSetTimer = TRUE;
    nTimerBaseTick = RalTimeGetTicks();
    nTimerTick = inMilliseconds;
    nTimerId = id;
    return TRUE;
}

bool RalTimeKillTimer(int id) {
    bSetTimer = FALSE;
    nTimerBaseTick = 0;
    nTimerTick = 0;
    nTimerId = -1;
    return TRUE;
}
```

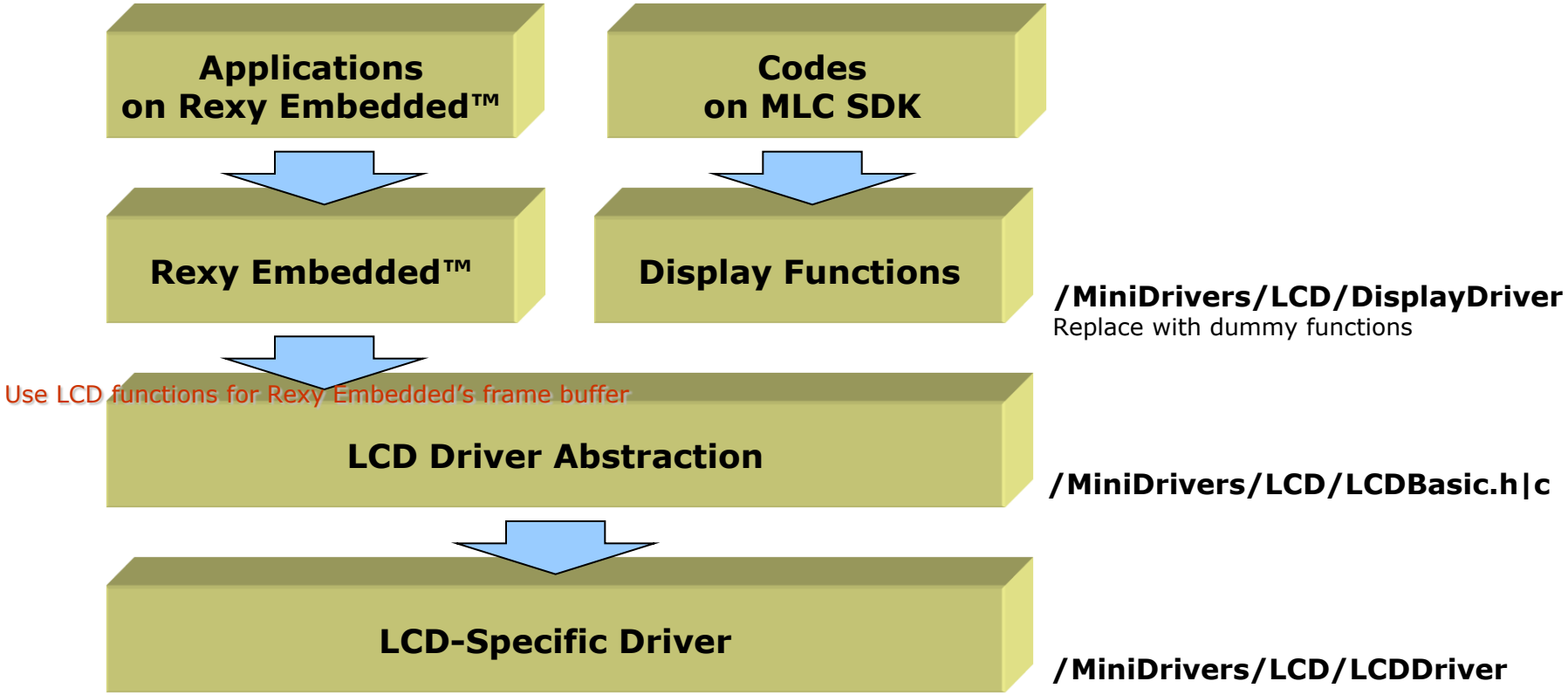
Timer Code in Rexy Embedded™

- Rexy_halinterface.c

```
ruint32 RalTimeGetTicks(void)
{
   return OSTimeGet() * 5;
}

bool RalEventGet(pmsg outMsg, ruint32 inTimeoutMs)
{
   // Timer Polling Routine
   if ((bSetTimer == TRUE) &&
      ((RalTimeGetTicks() - nTimerBaseTick) > nTimerTick))
   {
      outMsg->wmsg = AWM_TIMER;
      outMsg->wparam = nTimerId;
      outMsg->lparam = 0;
      return TRUE;
   }
   return FALSE;
}
```

Display Code in Rexy Embedded™



Display Code in Rexy Embedded™

- Rexy_halinterface.c

```
bool RalScreenFlush(void)
{
    int x, y;
    OEM_LcdDriverSetArea(0, 0, DEVICE_WIDTH - 1, DEVICE_HEIGHT - 1);
    for (x = 0; x < DEVICE_WIDTH; x++)
        for (y = 0; y < DEVICE_HEIGHT; y++)
            dLCD_DATAOUT(lcd_bmp_buffer[x][y]);
    return TRUE;
}
```

Display Code in Rexy Embedded™

- Rexy_halinterface.c

```
bool RalScreenFlushPart(rect* inBoundary)
{
    int x, y, sx, sy, ex, ey;
    sx = 0;    sy = 0;    ex = DEVICE_WIDTH -1;   ey = DEVICE_HEIGHT-1;
    if ( inBoundary ) {
        if ( inBoundary->left > sx ) sx = inBoundary->left;
        if ( inBoundary->right < ex ) ex = inBoundary->right;
        if ( inBoundary->top > sy ) sy = inBoundary->top;
        if ( inBoundary->bottom < ey ) ey = inBoundary->bottom;
    }
    else
        return FALSE;
    OEM_LcdDriverSetArea(sx, sy, ex, ey);
    for (x = sx; x <= ex; x++)
        for (y = sy; y <= ey; y++)
            dLCD_DATAOUT(lcd_bmp_buffer[x][y]);
    return TRUE;
}
```

Standard C Library in Rexy Embedded™

- rexy_clib.h/c
  - Memory Allocation Routine: Implemented using Rexy codes.
  - Unicode String Routine: Implemented from the scratch.
  - Other Functions: Use template codes.

External Resource in Rexy Embedded™

- /core/source/rexy_resource.c
  - Modify: Call the new HAL functions for resource.

- /hal/<target>/device/rexy_halinterface.c
  - Implement the HAL functions for resource.
    - Use MLC SDK's Overall codes. (Copy & Paste)
    - Add the new image type.
    - Support unified function for built-in and external resource on flash.

6. API REVIEW

1. Windows
2. Message
3. GDI
4. Timer
5. Control

Representative APIs among more than 160 powerful ones !

- Window

  WinCreateWindow / **WinDestroyWindow** / **WinShowWindow** / **WinUpdateWindow** / **WinInvalidateWindow**
  WinGetFirstControl / WinAddControl / WinDestroyControl / **WinSetSelection** / **WinGetSelection** /
  WinSetshowUpControl / WinSetShowDownControl / WinSetCurrentProc

- Message

  MsgGetMessage / MsgPeekMessage / MsgDispatchMessage / MsgDispatchSystemMessage / **MsgSendMessage**
  MsgPostMessage / MsgRemoveMessage / MsgPostQuitMessage / MsgRouteMessage / MsgDefWindowProc

- GDI

  GdiSetPixel / GdiDrawLine / GdiRectangle / **GdiFillRect** / **GdiTextOut** / GdiBitBlt / GdiStretchBlt / **GdiLoadBitmap**
  / **GdiDrawBitmap** / **GdiSetBkColor** / **GdiGetBkColor** / **GdiSetFgColor** / **GdiGetFgColor** / **GdiFlushAll** /
  GdiFlushPart / GdiBitmapCopy / GdiGetScreenInfo

- Timer

  UsrSetTimer / **UsrKillTimer** / UsrGetCurrentTime / UsrGetCurrentTicks

- Control

  CtlCreate / **CtlDestory** / **CtlDo** / **CtlSeti** / **CtlGeti** / **CtlSetPtr** / **CtlGetPtr**

WND WinCreateWindow (WINPROC wndproc, int x, int y, int w_width, int w_height, int mem)

- Description

  Create the panel(parent) window

  Child windows are created by calling the WinAddControl() function

- Parameter

  wndproc : Window procedure. All events related to the window are processed

  x : Left-topmost X-coordinate

  y : Left-topmost Y-coordinate

  w_width : Width of window

  w_height : Height of Window

  mem : Flag to specify the window as "memory window" or "screen window"

  - WND_MEM : Memory window
  - WND_SCREEN : Screen window

- Return value

  If function succeeds, returns the handle to new window. If function fails, returns NULL

void WinShowWindow (WND panel, int status)

- Description

  Show or hide window with the specified flag

  * status argument type

  1. SHOW_PANEL : Show the window
  2. HIDE_PANEL : Hide the window
  3. LEAVE_BKGND : Do not change the background.
  4. ERASE_BKGND : Send WM_ERASEBKGND message to the specified window.
  5. HOLD_FOCUS : Use with HIDE_PANEL flag to keep focus even though window gets hidden

- Parameter

  panel : Window handle to show or hide

  status : Specify the window status

Return value

None

int WinDestroyWindow(WND panel)

- Description

  Destroy the window

- Parameter

  panel : Window handle to destroy

- Return value

  Always returns 0

WND WinAddControl (WND panel, int type, int id, int style, rect * location, WINPROC wndproc)

- Description

  Add control window to panel window

- Parameter

  panel : Parent window. panel has new child control.

  type : Control type. Each basic controls have unique type.

  id : Control window ID

  style : Control window style

  location : Location information for new control window in parent window

  wndproc : Control procedure. All events related to the control are processed

- Return value

  If function succeeds, returns the handle to new control window. If function fails, returns NULL

int WinDestroyControl(WND panel)

- Description

  Destroy the control window

- Parameter

  panel : Control window handle to destroy

- Return value

  Always returns 0

int WinSetShowUpControl (WND panel)

- Description

  Change the panel's z-order up

- Parameter

  panel : Panel window handle

- Return value

  If succeed, returns 0

  Otherwise returns 1

int WinSetShowDownControl (WND panel)

- Description

  Change the panel's z-order down

- Parameter

  panel : Panel window handle

- Return value

  If succeed, returns 0

  Otherwise returns 1

API REVIEW

Message API

int MsgGetMessage (WND panel, pmsg msg)

- Description

  Get message from message queue

  If there is no message in queue, operation is blocked

- Parameter

  panel : Panel window handle

  msg : Message structure pointer

- Return value

  If retrieved message is WM_QUIT, returns 0

  Otherwise returns 1

int MsgPeekMessage (WND panel, pmsg msg)

- Description

  Retrieve message from message queue

  If no message is in the queue, return immediately

- Parameter

  panel : Panel window handle

  msg : Message structure pointer

- Return value

  If message is peeked successfully, returns 1

  Otherwise returns 0

int MsgDispatchMessage (pmsg msg)

- Description

  Send the message to appropriate window procedure
- Parameter

  msg : Message structure pointer
- Return value

  Depends on the message and window procedure. .

int MsgDispatchSystemMessage (pmsg msg)

- Description

  Dispatch the message to the system panel.
- Parameter

  msg : Message structure pointer.
- Return value

  If message is dispatched successfully, returns 1. Otherwise returns 0.

API REVIEW

Message API

int MsgSendMessage (WND panel, int Msg, int wParam, int lParam)

- Description

  Send the message to the window

  Gets blocked until sent message is processed. Process message by calling window procedure directly

- Parameter

  panel : Window handle

  Msg : Message type

  wParam : Message dependent data

  lParam : Message dependent data

- Return value

  Depends on the message and window procedure

Message API

int MsgPostMessage (WND panel, int Msg, int wParam, int lParam)

- Description

  Send message to the message queue

  Puts message in the message queue and return immediately

- Parameter

  panel : Window handle

  Msg : Message type

  wParam : Message dependent data

  lParam : Message dependent data

- Return value

  None

void GdiDrawLine(WND wnd, int x1, int y1, int x2, int y2)

- Description

  Draws a line

  1. Draws a line from (x1, y1) to (x2, y2) using current foreground color
  2. Does not draw the endpoint pixel
  3. It calls GdiDrawLineWidth() function using fixed value(1) by width argument

- Parameter

  wnd : Window to draw line

  x1 : X coordinate of line start point

  y1 : Y coordinate of line start point

  x2 : X coordinate of line endpoint

  y2 : Y coordinate of line endpoint

- Return value

  None

void GdiFillRect(WND wnd, rect * src, COLORVAL fill_color)

- Description

  Draws a rectangle with selected color

  1. Draws a rectangle in target window using selected fill_color
  2. Does not draw rightmost and bottommost border of rectangle

- Parameter

  wnd : Window to draw rectangle

  src : Structure of rectangle to draw. Coordinates are followed by target window

  fill_color : Color to fill rectangle

- Return value

  void

int GdiTextOut(WND wnd, char * string, int length, int x, int y)

- Description

  Writes character string at the specified location in the window

- Parameter

  wnd : Window to draw characters

  string : Points to the character string to be drawn

  length : Length in bytes of characters to be drawn

  x : X coordinate of draw(left bound)

  y : Y coordinate of draw(top bound)

- Return value

  Always return 0

void GdiFlushAll(WND wnd)

- Description

  Transfer bitmap image of RexyE video buffer to LCD device.

  RexyE has internal video buffer in non-cached memory 0x0CFDA800. Updating this memory does not update LCD screen immediately. To update LCD screen, you must use this function or GdiFlushPart. This functions implements weak mutual exclusion, it will work if your tasks use only this function to update screen.

- Parameter

  wnd : Window handle which has video memory to be transferred as structure member ‘dev’

- Return value

  None

void GdiFlushPart(WND wnd, rect *boundary)

- Description

  Transfer bitmap image of RexyE video buffer to LCD device.

  It does same thing as GdiFlushAll. But, with this function, you can specify boundary to be updated.

- Parameter

  wnd : Window handle which has video memory to be transferred as structure member ‘dev’

  boundary : The area to be update in video memory

- Return value

  None

API REVIEW

Timer API

int UsrSetTimer(WND window, int nIDEvent, int uElapse)

- Description

  Set the timer in msec.

- Parameter

  window : Window handle to set the timer

  nIDEvent : Timer ID. Timer ID should be larger than REXY_APPLICATION_TIMER

  uElapse : Time elapse. Unit is msec

- Return value

  If timer set succeeds, returns Timer ID. Otherwise returns error code

int UsrKillTimer(WND window, int uIDEvent)

- Description

  Kill the specified timer.

- Parameter

  window : Window handle to kill timer

  uIDEvent : Timer ID. Timer ID should be larger than REXY_APPLICATION_TIMER

- Return value

  If timer is killed successfully, returns 1. Otherwise returns 0

API REVIEW

Timer API

int UsrGetCurrentTime(struct tm * current_time)

- Description

  Get current time.

- Parameter

  current_time : A structure that will be received current time.

- Return value

  Success : RxErrNone

  Error : error code

unsigned long UsrGetCurrentTicks(void)

- Description

  Returns the current tick value as unsigned integer.

- Parameter

  None

- Return value

  return current tick value

WND CtlCreate(WND wnd, int nType, rect *location, CTL_CB_PROC pfnProc, int *pErr)

- Description

  Create control.

- Parameter

  wnd : Parent window

  nType : Control type

  location : Position of Control window

  pfnProc : Callback fuction

  pErr : Error pointer

- Return value

  If control create succeeds, returns new created control window pointer. Otherwise returns NULL.

void CtlDestroy(WND wnd, int *pErr)

- Description

  Destroy control

- Parameter

  wnd : Control instance window pointer to destroy

  pErr : Error pointer

- Return value

  NONE

void CtlSeti(WND wnd, int nCommand, int iValue, int *pErr)

- Description

  Set control property value

- Parameter

  wnd : Control window pointer

  nCommand : Control command

  iValue : Data to set

  pErr : Error pointer

- Return value

  NONE

int CtlGeti(WND wnd, int nCommand, int *pErr)

- Description

  Get control property value

- Parameter

  wnd : Control window pointer

  nCommand : Control command

  pErr : Error pointer

- Return value

  Returns control property value

void CtlSetPtr(WND wnd, int nCommand, void *pValue, int *pErr)

- Description

  Set control property value pointer (function, structure data, etc…)

- Parameter

  wnd : Control window pointer

  nCommand : Control command

  pValue : Data pointer to set

  pErr : Error pointer

- Return value

  NONE

void * CtlGetPtr(WND wnd, int nCommand, int *pErr)

- Description

  Get control property value pointer (function, structure data, etc…)

- Parameter

  wnd : Control window pointer

  nCommand : Control command

  pErr : Error pointer

- Return value

  Returns control property value pointer

void CtlSetiPos(WND wnd, int nCommand, int iPos, int iValue, int *pErr)

- Description

  Set the control property with position value

- Parameter

  wnd : Control window pointer
  nCommand : Control command
  iPos : Position value
  iValue : Setting value
  pErr : Error pointer

- Return value

  NONE

int CtlGetiPos(WND wnd, int nCommand, int iPos, int *pErr)

- Description

  Get the control property with position value

- Parameter

  wnd : Control window pointer
  nCommand : Control command
  iPos : Position value
  pErr : Error pointer

- Return value

  Returns control property value

void CtlSetiPtr(WND wnd, int nCommand, int iPos, void *pValue, int *pErr)

- Description

  Set the pointer of the control property value

- Parameter

  wnd : Control window pointer
  nCommand : Control command
  iPos : Position value
  pValue : Data pointer to set
  pErr : Error pointer

- Return value

  NONE

void * CtlGetiPtr(WND wnd, int nCommand, int iPos, int *pErr)

- Description

  Get the pointer of the control property value

- Parameter

  wnd : Control window pointer
  nCommand : Control command
  iPos : Position value
  pErr : Error pointer

- Return value

  Return the pointer of the control property value

int CtlDo(WND wnd, int nCommand, int wParam, int lParam, int *pErr)

- Description

  Do control command process

- Parameter

  wnd : Control window pointer

  nCommand : Control command

  wParam : parameter for data

  lParam : parameter for data

  pErr : Error pointer

- Return value

  Returns control property value

List Control Command Type 1

- You can set the List Control property through Command Procedure

ITEM	Command Type	Type	Description
Property SET/GET	CTL_LIST_CMD_SET_ITEMNUM	int	Set list item number
	CTL_LIST_CMD_GET_ITEMNUM		Get item number
	CTL_LIST_CMD_SET_TOPINDEX	int	Set top index
	CTL_LIST_CMD_GET_TOPINDEX		Get top index
	CTL_LIST_CMD_SET_FOCUS	int	Set focus item
	CTL_LIST_CMD_GET_FOCUS		Get focus item
	CTL_LIST_CMD_SET_FONT	int	Set font
	CTL_LIST_CMD_SET_LINE_HEIGHT	int	Set line height
	CTL_LIST_CMD_SET_CTRL_BGCOLOR	int	Set the background color of the control
	CTL_LIST_CMD_SET_COLUMN_MARGIN	int, rect *1	Set the column margin
	CTL_LIST_CMD_SET_CTRL_POS	rect *1	Set control window position
	CTL_LIST_CMD_SET_LINE_INTERVAL	int	Set the line interval between items

List Control Command Type 2

- You can set the List Control property through Command Procedure

ITEM	Command Type	Type	Description
Property SET/GET	CTL_LIST_CMD_SET_USE_TEXTSCROLL	int	Set whether text scrolling is used or not
	CTL_LIST_CMD_SET_COLUMN_NUM	int	Set column number
	CTL_LIST_CMD_SET_COLUMN_POS	int, int	Set column position
	CTL_LIST_CMD_SET_TEXT_ALIGN	int, int	Set the text alignment of the column
	CTL_LIST_CMD_GET_LINENUM	int	Get line number
Callback SET/GET	CTL_LIST_CMD_SET_DRAW_ITEM_FOCUS_CB	CtlListDrawItemFocusCB*2	Set the callback function of item focus drawing
	CTL_LIST_CMD_SET_DRAW_COLUMN_CB	CtlListDrawColumnCB*2	Set the callback function of column drawing
	CTL_LIST_CMD_SET_DRAW_KEY_DOWN_CB	CtlListProcessKeyDownCB*2	Set the callback function of key down drawing
	CTL_LIST_CMD_SET_DRAWCBDATA	void *	Set data callback function pointer
Do	CTL_LIST_CMD_DO_DRAW	None	Execute control drawing totally
	CTL_LIST_CMD_DO_ITEMDRAW	int	Execute item drawing

List Control Command Type

1. rect *

```
typedef struct _rect {
    int left;
    int top;
    int right;
    int bottom;
} rect;
```

2. CtlListDrawItemFocusCB

```
typedef int (*CtlListDrawItemFocusCB)(WND hWnd, int nLine, void *pData);
```

3. CtlListDrawColumnCB

```
typedef int (*CtlListDrawColumnCB)(WND hWnd, int line, int column, void* pData);
```

4. CtlListProcessKeyDownCB

```
typedef int (*CtlListProcessKeyDownCB)(WND hWnd, int nModifier, void *pData);
```

Popup Control Command Type

- You can set the Popup Control property through Command Procedure

Item	Command Type	Type	Description
Property SET/GET	CTL_POPUP_CMD_SET_TYPE	ruint16	Set control style
	CTL_POPUP_CMD_SET_TOP	ruint16	Set top position
	CTL_POPUP_CMD_SET_LEFT	ruint16	Set left position
	CTL_POPUP_CMD_SET_RIGHT	ruint16	Set right position
	CTL_POPUP_CMD_SET_BOTTOM	ruint16	Set bottom position
	CTL_POPUP_CMD_SET_FGCOLOR	COLORVAL	Set foreground color
	CTL_POPUP_CMD_SET_BGCOLOR	COLORVAL	Set background color
	CTL_POPUP_CMD_SET_BOXCOLOR	COLORVAL	Set bound rectangle color
	CTL_POPUP_CMD_SET_TIMEOUT	int	Set timeout setting
Callback SET/GET	CTL_POPUP_CMD_SET_CB_DRAW	CtlPopupDrawCB[1]	Set draw callback function pointer
	CTL_POPUP_CMD_SET_CB_KEYDOWN	CtlPopupKeyDownDB[2]	Set key down callback function pointer
Do	CTL_POPUP_CMD_DO_DRAW	None	Execute control drawing totally

Popup Control Command Type

1. CtlPopupDrawCB

```
typedef int (*CtlSlideTextDrawCB)(WND hWnd);
```

2. CtlPopupKeyDownCB

```
typedef int (*CtlSlideTextKeyDownCB)(WND hWnd, int wParam);
```

Slide Text Control Command Type

- You can set the Slide Text Control property through Command Procedure

Item	Command Type	Type	Description
Property SET/GET	CTL_SLIDETEXT_CMD_SET_X	rint16	Set X coordinate of the text
	CTL_SLIDETEXT_CMD_SET_Y	rint16	Set Y coordinate of the text
	CTL_SLIDETEXT_CMD_SET_FGCOLOR	COLORVAL	Set foreground color of the text
	CTL_SLIDETEXT_CMD_SET_TOP	rint16	Set top position of the win rect
	CTL_SLIDETEXT_CMD_SET_LEFT	rint16	Set left position of the win rect
	CTL_SLIDETEXT_CMD_SET_RIGHT	rint16	Set right position of the win rect
	CTL_SLIDETEXT_CMD_SET_BOTTOM	rint16	Set bottom position of the win rect
	CTL_SLIDETEXT_CMD_SET_BGCOLOR	ruint16	Set background color of the win rect
	CTL_SLIDETEXT_CMD_SET_WNDRECT	rect *1	Set win rect at the same time
	CTL_SLIDETEXT_CMD_SET_TIMEOUT	ruint16	Set time interval for next step
	CTL_SLIDETEXT_CMD_SET_MAXWIDTH	ruint16	Set max width in width of character to be slide
	CTL_SLIDETEXT_CMD_SET_DATA	void *	Set string data pointer
Callback SET/GET	CTL_SLIDETEXT_CMD_SET_CB_DRAW	CtlSlidetextDrawCB2	Set draw callback function pointer
	CTL_SLIDETEXT_CMD_SET_CB_KEYDOWN	CtlSlideTextKeyDownCB3	Set key down callback function pointer

Slide Text Control Command Type

- You can set the Slide Text Control property through Command Procedure

	Command Type		Description
Do	CTL_SLIDETEXT_CMD_DO_LFRONT	None	Execute shift a text to left side
	CTL_SLIDETEXT_CMD_DO_LROUND	None	Execute shift a text to round
	CTL_SLIDETEXT_CMD_DO_PLAY	None	Execute text slide play
	CTL_SLIDETEXT_CMD_DO_STOP	None	Execute text slide stop
	CTL_SLIDETEXT_CMD_DO_PAUSE	None	Execute text slide pause
	CTL_SLIDETEXT_CMD_DO_START_FRONT	None	Execute starting in front of the win rect
	CTL_SLIDETEXT_CMD_DO_START_END	None	Execute starting in end of the win rect
	CTL_SLIDETEXT_CMD_DO_DRAW	None	Execute control drawing totally

Slide Text Control Command Type

1. rect *

```
typedef struct _rect {
    int left;
    int top;
    int right;
    int bottom;
} rect;
```

2. CtlSlideTextDrawCB

```
typedef int (*CtlSlideTextDrawCB)(WND hWnd, COLORVAL nFgColor, COLORVAL nBgColor);
```

3. CtlSlideTextKeyDownCB

```
typedef int (*CtlSlideTextKeyDownCB)(WND hWnd, int wParam);
```

Animation Bitmap Control Command Type

- You can set the Animation Bitmap Control property through Command Procedure

Item	Command Type	Type	Description
Property SET/GET	CTL_ANIBMP_CMD_SET_TOP	rint16	Set top position of the win rect
	CTL_ANIBMP_CMD_SET_LEFT	rint16	Set left position of the win rect
	CTL_ANIBMP_CMD_SET_RIGHT	rint16	Set right position of the win rect
	CTL_ANIBMP_CMD_SET_BOTTOM	rint16	Set bottom position of the win rect
	CTL_ANIBMP_CMD_SET_WINDRECT	rect *1	Set win rect at the same time
	CTL_ANIBMP_CMD_SET_BGCOLOR	COLORVAL	Set background color of the win rect
	CTL_ANIBMP_CMD_SET_FRAME_NO	uint16	Set image count number
	CTL_ANIBMP_CMD_SET_P_IDS	int *	Set image resource ID pointer
	CTL_ANIBMP_CMD_SET_PLAYTYPE	enum2	Set play mode type
	CTL_ANIBMP_CMD_SET_TIMEOUT	int	Set time interval for next setp
	CTL_ANIBMP_CMD_SET_P_TIMEOUT	Int *	Set time interval array pointer
	CTL_ANIBMP_CMD_SET_START_STOP	bool	Set animation bitmap start/stop flags
Callback SET/GET	CTL_ANIBMP_CMD_SET_CB_KEYDOWN	CtlAniBmpKeyDownCB3	Set Key down callback function pointer

Animation Bitmap Control Command Type

1. rect *

```
typedef struct _rect {
    int left;
    int top;
    int right;
    int bottom;
} rect;
```

2. enum

```
enum
{
    CTL_ANIBMP_PLAY_NORMAL,
    CTL_ANIBMP_PLAY_ROTATE,
    CTL_ANIBMP_PLAY_REVERSE,
    CTL_ANIBMP_PLAY_REVERSE_ROTATE,
    CTL_ANIBMP_PLAY_RANDOM
};
```

3. CtlAniBmpKeyDownCB

```
typedef int (*CtlAniBmpKeyDownCB)(WND hWnd, int wParam, int *pErr);
```

Edit Control Command Type

- You can set the Edit Control property through Command Procedure

Item	Command Type	Type	Description
Property SET/GET	CTL_EDIT_CMD_SET_WND_RECT	rect *1	Set win rect at the same time
	CTL_EDIT_CMD_GET_WND_RECT		Get win rect at the same time
	CTL_EDIT_CMD_SET_STRING	void *	Set new string data pointer
	CTL_EDIT_CMD_GET_STRING		Get current string data pointer
	CTL_EDIT_CMD_SET_MAX_LEN	int	Set max length for the edit
	CTL_EDIT_CMD_GET_MAX_LEN		Get current max length
	CTL_EDIT_CMD_SET_PASSWORD	bool	Set new password mode flags
	CTL_EDIT_CMD_GET_PASSWORD		Get current password mode flags
	CTL_EDIT_CMD_SET_READONLY	bool	Set new read only mode flags
	CTL_EDIT_CMD_GET_REAYONLY		Get current read only mode
	CTL_EDIT_CMD_SET_CURSOR_INDEX	int	Set new caret position in edit box
	CTL_EDIT_CMD_GET_CURSOR_INDEX		Get current caret position in edit box
	CTL_EDIT_CMD_SET_MULTILINE	bool	Set new multi line mode flags
	CTL_EDIT_CMD_GET_MULTILINE		Get current multi line mode flags

Edit Control Command Type

- You can set the Edit Control property through Command Procedure

Item	Command Type	Type	Description
Callback SET/GET	CTL_EDIT_CMD_SET_CB_DRAW	CtlEditCBDraw[2]	Set draw callback function pointer
	CTL_EDIT_CMD_GET_CB_DRAW		Get draw callback function pointer
	CTL_EDIT_CMD_SET_CB_KEYDOWN	CtlEditCBKeyDown[3]	Set key down callback function pointer
	CTL_EDIT_CMD_GET_CB_KEYDOWN		Get key down callback function pointer
Do	CTL_EDIT_CMD_DO_DRAW	None	Execute control drawing totally

Edit Control Command Type

1. rect *

```
typedef struct _rect {
    int left;
    int top;
    int right;
    int bottom;
} rect;
```

2. CtlEditCBDraw

```
typedef int (*CtlEditCBDraw)(WND hWnd, void *pString, int nLength);
```

3. CtlEditCBKeyDown

```
typedef int (*CtlEditCBKeyDown)(WND hWnd, int nKeyCode, void *pString, int *pLength);
```

Reference Type Info

1. typedef

```
typedef unsigned short ruint16;
typedef signed short rint16;
typedef unsigned char ruint8;
typedef ruint8 bool;
typedef ruint16 COLORVAL;
```

2. rect *

```
typedef struct _rect {
    int left;
    int top;
    int right;
    int bottom;
} rect;
```